Spring 1998

A low-cost high-speed twin-prefetching DSP-based shared-memory system for real-time image processing applications

Charalambos Stephanou Christou
New Jersey Institute of Technology

ABSTRACT

A LOW-COST HIGH-SPEED TWIN-PREFETCHING DSP-BASED SHARED-MEMORY SYSTEM FOR REAL-TIME IMAGE PROCESSING APPLICATIONS

by
Charalambos Stephanou Christou

This dissertation introduces, investigates, and evaluates a low-cost high-speed twin-prefetching DSP-based bus-interconnected shared-memory system for real-time image processing applications. The proposed architecture can effectively support 32 DSPs in contrast to a maximum of 4 DSPs supported by existing DSP-based bus-interconnected systems. This significant enhancement is achieved by introducing two small programmable fast memories (Twins) between the processor and the shared bus interconnect. While one memory is transferring data from/to the shared memory, the other is supplying the core processor with data. The elimination of the traditional direct linkage of the shared bus and processor data bus makes feasible the utilization of a wider shared bus i.e., shared bus width becomes independent of the data bus width of the processors. The fast prefetching memories and the wider shared bus provide additional bus bandwidth into the system, which eliminates large memory latencies; such memory latencies constitute the major drawback for the performance of shared-memory multiprocessors. Furthermore, in contrast to existing DSP-based uniprocessor or multiprocessor systems the proposed architecture does not require all data to be placed on on-chip or off-chip expensive fast memory in order to reach or maintain peak

performance. Further, it can maintain peak performance regardless of whether the processed image is small or large.

The performance of the proposed architecture has been extensively investigated executing computationally intensive applications such as real-time high-resolution image processing. The effect of a wide variety of hardware design parameters on performance has been examined. More specifically tables and graphs comprehensively analyze the performance of 1, 2, 4, 8, 16, 32 and 64 DSP-based systems, for a wide variety of shared data interconnect widths such as 32, 64, 128, 256 and 512. In addition, the effect of the wide variance of temporal and spatial locality (present in different applications) on the multiprocessor's execution time is investigated and analyzed. Finally, the prefetching cache-size was varied from a few kilobytes to 4 Mbytes and the corresponding effect on the execution time was investigated. Our performance analysis has clearly showed that the execution time converges to a shallow minimum i.e., it is not sensitive to the size of the prefetching cache. The significance of this observation is that near optimum performance can be achieved with a small (16 to 300 Kbytes) amount of prefetching cache.

A LOW-COST HIGH-SPEED TWIN-PREFETCHING DSP-BASED SHARED-MEMORY SYSTEM FOR REAL-TIME IMAGE PROCESSING APPLICATIONS

by
Charalambos Stephanou Christou

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

May 1998

APPROVAL PAGE

A LOW-COST HIGH-SPEED TWIN-PREFETCHING DSP-BASED SHARED-MEMORY SYSTEM FOR REAL-TIME IMAGE PROCESSING APPLICATIONS

Charalambos Stephanou Christou

2/2/98

Dr. Constantine N. Manikopoulos, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

2/2/1998

Dr. Dennis Karvelas, Committee Member — Date
Director of M.S. in Telecommunications, Computer Science Dept., NJIT

2/2/98

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

2/2/98

Dr. John D. Carpinelli, Committee Member — Date
Associate Professor of Electrical and Computer Engineering and Computer and Information Science, NJIT

2/2/98

Dr. Slawomir Piatek, Committee Member — Date
Special Lecturer, Physics Department, NJIT

BIOGRAPHICAL SKETCH

Author: Charalambos Stephanou. Christou

Degree: Doctor of Philosophy in Electrical Engineering

Date: May 1998

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1998
- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1988
- Bachelor of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1987

Major: Electrical Engineering

Publications:

1. Charalambos S. Christou and Constantine N. Manikopoulos, "Wider Shared-Bus - A Solution for Increasing the Number of Effectively Supported Processors in a Multiprocessor System," to be submitted for publication.

2. Charalambos S. Christou and Constantine N. Manikopoulos, "A Low-Cost DSP-Based Shared-Memory Multiprocessor System for Real-Time Applications," to be submitted for publication

3. Charalambos S. Christou and John D. Carpinelli, "Determination of Optimal Cache Size through Modeling and Simulation," *Proceed. of Twentieth Pittsburgh Conference on Modeling and Simulation,* pp. 1267-1271, May 1989.

This dissertation is dedicated to
my parents Stephanos and Evagelia,
my wife Filanthi, my sister Anthoula,
and my brothers Andreas and Christos

ACKNOWLEDGMENT

I would like to express my gratitude to my advisor, Dr. Constantine N. Manikopoulos, for his guidance throughout the course of the dissertation research.

Special thanks to Dr. Dennis Karvelas, Dr. Edwin Hou, Dr. John D. Carpinelli, and Dr. Slawomir Piatek for serving as members of the dissertation committee.

Special thanks to my friends Socrates Ioannides, Tasos Kondos, Ioannis Tzathas, Lazaros Vastardos, Vaggelis Tsimis, Christos Banias, Michalis and Nancy Nikolaou, Antonis Tjirkallis, Jim Koroniades, Evzonas Andreas, Akis Kleanthous, Michalis Sideras, Christos Sideras, Giannis Milonas, Nikos Antoniou, Kyriakos Mouskos, Giorgos Antoniou, Giorgos Donos, Bambos and Maria Pafiti, Lefteris and Andri Panagiotou, Kostas and Maria Tsatsos, Panagiotis and Maria Vastardos, Kyriakos and Themi Vyras, and Petros and Andri Kyriakou for their love and enduring support.

Finally, I would like to thank very much my wife Filanthi, my mother Evagelia, my father Stefanos, my sister Anthoula, and my brothers Andreas and Christos for their spiritual support throughout my academic endeavors. I can hardly find words to express my gratitude for their love and encouragement through the years.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(continued)

Chapter Page

TABLE OF CONTENTS
(continued)

Chapter Page

LIST OF TABLES

Table **Page**

LIST OF TABLES
(continued)

Table **Page**

LIST OF TABLES
(continued)

Table **Page**

LIST OF TABLES
(continued)

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(continued)

Figure **Page**

LIST OF FIGURES
(continued)

Figure **Page**

LIST OF FIGURES
(continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Parallel processing has had a tremendous impact on many areas of computer application. High raw computing power of parallel computers now can meet application requirements that were until recently beyond the capability of conventional computing techniques. One of the key problems to be solved in the area of parallel processing is the bridging of the gap between processors' speed and memory latency. Processor performance has increased dramatically over the past few years while memory latency and bandwidth have progressed at a much slower pace. Large latencies have considerably reduced the number of processors, which can be effectively supported in shared memory parallel computers. The focus of this dissertation is a new cost-effective parallel computer architecture that reduces memory latency and effectively supports a greater number of processing elements. This chapter provides an introductory background in this fast growing research area. The outline of the dissertation as well as the motivations, objectives, and contributions are presented at the end of the chapter.

1.1 Parallel Processing Systems

A parallel computer, in general, has attributes such as the number of processors (or number of nodes), the memory system, peripherals, and the interconnection network (a collection of wires and connectors for data transactions among processors, memory modules, and peripheral devices). Parallel architectures can be classified in two large

classes: *shared-memory multiprocessors* and *message-passing multicomputers* [32][31]. Multiprocessors have a single, global, shared address space visible to all processors. Any processor can read or write any word in the address space by moving data from or to a memory address. Communication is via the shared memory. Multicomputers do not have a shared memory and must communicate by message passing.

1.1.1 Shared-Memory Multiprocessors

Multiprocessors are also called *tightly coupled systems* due to the high degree of resource sharing. Three shared-memory multiprocessor models are primarily used: The uniform-memory-access (UMA) model, the nonuniform-memory-access (NUMA) model and the cache-only-memory access (COMA) model [32]. They differ in the way the memory and other resources are distributed. If the time taken by a processor to access any memory word in the system is identical the computer is classified as UMA (model shown in

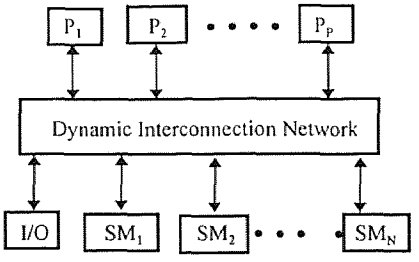

Figure 1.1 The UMA multiprocessor model

Figure 1.1). All processors have equal access time to all memory locations in all shared-memory modules (marked as SM) under the condition of no network congestion. That is why it is called a uniform-memory-access model. In the NUMA model in Figure 1.2,

however, the shared memory is physically distributed to all processors (called local memories). The collection of all local memories forms a global address space accessible to all processors. The time to access a remote memory bank is longer than the time to access a local one (labeled as LM), because a processor has to go through the interconnection network when accessing the former. The cache-only-memory access (COMA) [28] is a special case of NUMA machine, in which the distributed main memories are converted to caches.

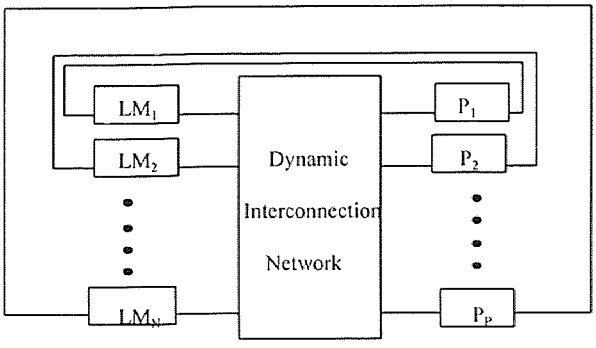

Figure 1.2 The NUMA multiprocessor model

Multiprocessor systems are suitable for general-purpose applications where programmability is the major concern. They preserve the intuitively appealing programming model provided by a single, linear memory address space. It is the opinion of many researchers [32][33][34] in the area of parallel processing that the recent failures of many parallel computing companies are partly due to the difficulty of programming and lack of software of their supercomputers. Mainly because of programming

complexities of message-passing multicomputers, the architectural trend for future general-purpose computers is in favor of shared-memory systems [32][33][37]. Another significant factor for the observed failure is the price/performance ratio. In fact, Ted Lewis states in [35] the following: "I could never understand why a 30-processor multicomputer costs as much as 130 workstations when both contain the same commodity chips." It should be also mentioned that price/performance ratio is always a major factor for a successful uniproccessor or multiprocessor system design.

Latency tolerance for memory access is a major limitation in shared-memory systems [3][4][13] due to the bandwidth constraint. For instance, four to eight modern processors (with private caches-with or without prefetching), accessing the same memory module, can easily use up all the available bandwidth [5][37].

1.1.1.1 Caches: Private caches in conjunction with hardware-based cache coherence maintenance [36] have contributed to the reduction of the ill effects of large memory access times. Caches are placed between the (fast) processor and (slow) main (shared) memory and have the basic function to hold regions of recently referenced shared-memory. References satisfied by the cache (called cache hits) proceed at processor speed; those unsatisfied (called misses) incur a cache miss penalty to fetch the corresponding data from the shared memory. Most modern processors must wait, or stall, until the data arrive. Caches increase system performance not only because data is available to the processor faster but also because they reduce congestion on the interconnection network, i.e., a high cache hit rate eliminates some of the traffic that would have otherwise gone out across the interconnection network to the shared-memory. Reduced traffic increases

the number of processors that can be effectively supported on the interconnection network. Caches explain the fact that multiprocessors systems are designed with less memory bandwidth than the one that would be required by the sum of the individual CPUs. When planning the design of a multiprocessor system, great consideration should be given to what kind of applications will be executed on it. The maximum speedup of an application is bounded by the speed of the interconnection network.

1.1.1.2 Dynamic Interconnection Networks: Multiple processors in shared-memory systems communicate and access the common memory through a dynamic interconnection network. This network implements all communication patterns based on program demands using switches and arbiters along the connecting paths to provide the dynamic connectivity. In increasing order of cost and performance, dynamic connection networks include bus systems, multistage interconnection networks (MIN), and crossbar switch networks. The performance is indicated by the network bandwidth, data transfer rate, network latency, and communication patterns supported.

The simplest and least costly way to construct a multiprocessor system is to connect the processors on a shared bus. In the past, commercial releases of bus based multiprocessors supported as many as 32 processors. The advent of high-performance ultra-fast processors has reduced that number to four [11] or eight [7]. For instance, each node of the Stanford DASH multiprocessor [14] is a bus-based cluster and supports only four high performance RISC processors. The amount of data which a shared bus can deliver to a computer system depends on its speed (clock rate), memory access time, and data-bus width. Conventional designs of shared-bus shared memory systems, naturally set

the width of the bus to the same size of the data bus width of the processor. The shared bus can only handle one transaction at a time, employing a single source; therefore limiting the amount of total data transferred per transaction to the data bus width of the processor. The proposed twin-prefetching system separates the traditional linkage of shared bus and processor bus and enables the utilization of a wider shared bus. A wider shared bus pumps into the system additional bandwidth and supports a greater number of processors. It makes the system scalable (unheard of for shared-bus shared memory system) since the width of the shared bus becomes independent from the processor bus width, i.e., bandwidth is increased by increasing the shared bus width.

The crossbar and multistage networks are more complex and provide higher bandwidth for higher cost. They are to be used if target performance cannot be achieved through bus interconnect [10]. Digital signal processing (DSP-based) parallel computers built for image processing [21][25][27] have no other choice but to employ crossbar and multistage switches in order to cope with the immense processing load.

1.1.1.3 Prefetching: Recent research has shown that prefetching in caches further reduces memory latencies and increases system performance [1][2][3][4][8][9][30]. Any prefetching scheme has as a goal to reduce the processor stall time by bringing data into the cache before they are referenced. Prefetching approaches proposed in the literature are software or hardware based. Software controlled prefetching schemes rely on the programmer/compiler to insert prefetch instructions prior to the instructions that trigger a miss. Hardware controlled prefetching schemes detect accesses with regular patterns and issue prefetches at run time. T. Mowry and A. Gupta [3] (software-controlled

prefetching), for example, report as much as 150% of performance improvement when applications with regular data access patterns are executed on the Stanford DASH multiprocessor [14]. Fredrik et. al. [4] report a 78% reduction in read miss by employing a simple hardware controlled prefetching technique which relies on an automatic prefetch of multiple consecutive blocks that follow the one that caused the miss in the cache. They are exploiting spatial locality of data.

1.1.2 Message-Passing Multicomputers

A message-passing multicomputer consists of multiple nodes interconnected by a point to point network. Each node is an autonomous computer including a processor, a private local memory, and possibly disks or I/O peripherals, as modeled in Figure 1.3. Internode communication is carried out by passing messages through the network while observing certain communication protocols. Such actions may involve multiple links (i.e., physical connections between nodes) and nodes, if the source is not directly connected to the destination.

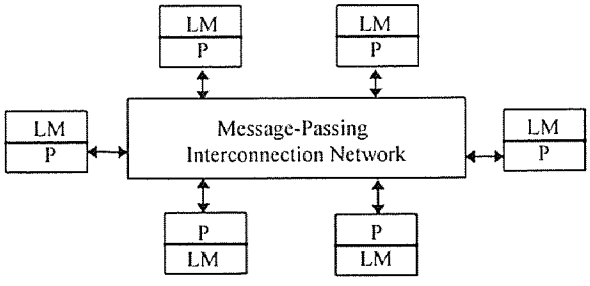

Figure 1.3 A multicomputer system

Some common network topologies in constructing interconnection networks for multicomputers are, as shown in Figure 1.4, binary tree, star, ring, mesh, hypercube, etc. They are also called static connection networks because all links between nodes are fixed after a network is built.

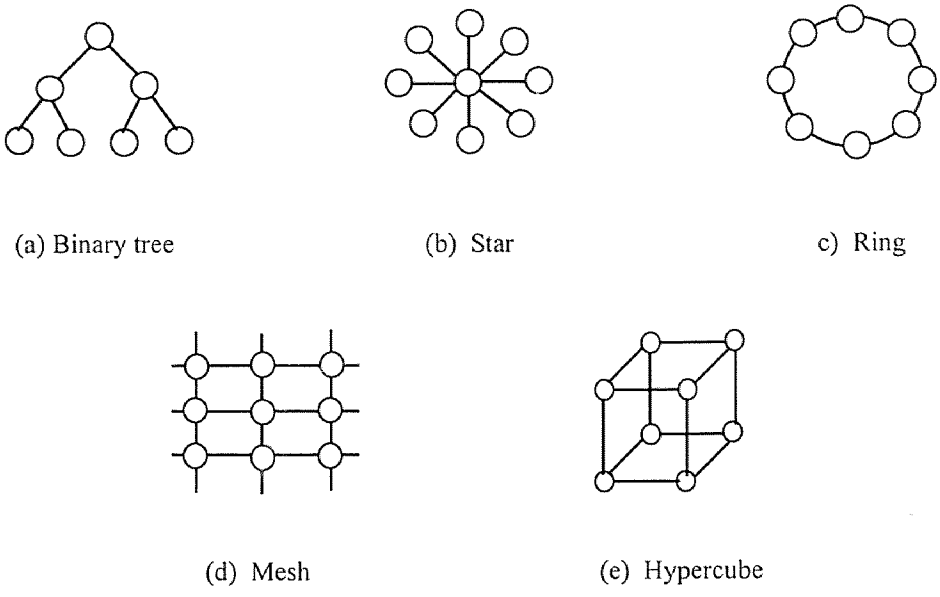

Figure 1.4 Common topologies for interconnection networks

Multicomputers achieve better scalable performance due to their distributed processor/memory nodes. However, message passing imposes a hardship on programmers to distribute the computations and data sets over the nodes or to establish efficient communication among nodes. Until intelligent compilers and efficient distributed operating systems become available, multicomputers will continue to lack programmability. Chandra et. al. [6] studied the strengths and weaknesses of the two fundamental mechanisms of message-passing and shared-memory by comparing the

performance of equivalent, well-written message-passing and shared-memory programs running on similar hardware. Each application program was produced in two versions and its performance was measured on closely-related simulators of a message-passing and a shared-memory machine. They found that three of the four shared-memory programs ran at roughly the same speed as their message-passing equivalents, even though their communication patterns were different. Similar results are reported in [12]. Therefore, if both paradigms achieve the same speedup it is preferable to choose the shared-memory approach which is more user friendly.

1.1.3 Image Processing

Parallel image processing and computer vision have exhibited a tremendous growth in the past decade. This process has been driven not only by the need for fast processing but also from the fact that parallelism suits well to the tasks of digital image processing and to the nature of digital images. Digital images are sampled on a rectangular grid and are stored on a two dimensional array [23]. Therefore, they possess an inherent geometrical parallelism [24]. This parallelism can be exploited by using a large two-dimensional array of processors, possibly one per image pixel. However, this is possible only for small images. Thus, a 512x512 or a 1024x1024 image is segmented (partitioned) in square blocks or in strips and each block/strip is assigned to a specific processor. The latter method allows general purpose parallel computers to solve problems in digital image processing.

Parallel architectures can be classified into two large classes: Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) machines. SIMD

machines dominated in parallel digital image processing in the past. Currently MIMD computers are progressively taking their space. The main reason is the wide availability of general purpose MIMD computers and their use by mainstream scientists and engineers. Some of these machines (e.g., DSP-based) are attractive because they combine low cost and high numerical performance [21][22][26]. MIMD machines are further divided into multiprocessors and multicomputers.

Many digital image processing algorithms are essentially local neighborhood operations in the form:

$$y_{ij} = F(x_{i+r,j+s}) \qquad (r,s) \in A \tag{1.1}$$

where x_{ij}, y_{ij} are the input and the output image respectively. F is an operator (linear or nonlinear) and A is its template window. The most frequently used window is square of size 3x3. It contains the pixels having city block distance 1,2 from the central pixel. Neighborhood parallelism denotes the possible parallel execution of local neighborhood operations. Local neighborhood operations are not usually executed in parallel in general-purpose parallel computers. They affect significantly the overall speedup of a parallel system due to the greater locality of data (spatial and temporal), which is observed when there is a greater the number of local neighborhood operations. It should be noted that data locality is a determining factor for the cache or prefetching cache hit ratio. Spatial locality means that if a location in memory is accessed, then others in the neighborhood will probably be accessed shortly. Temporal locality means that recently referenced items are likely to be referenced again in the near future [32].

1.2 Motivation, Objectives, and Contributions

Access to the common memory is the key limiting factor in the performance of shared-memory multiprocessors. The traversal of the processor-shared-memory interconnect employs large latencies as the number of processors increases. The advent of ultra-fast, heavily pipelined, multifunction, one-cycle-per-instruction-execution-time modern processors have made the problem worse [19], allowing only a handful of nodes to be effectively supported on a shared-bus shared-memory multiprocessor system. Moreover due to the significant memory requirements of the computationally intensive nature of digital image processing applications [18] DSP-based multiprocessors can support even a smaller number of processing units. This limitation enforces current commercial DSP-based shared-memory systems to employ expensive and complex crossbar switch networks in order to support even the small number of four high-performance DSPs [20][21]. Our vision is to meet the requirements for real-time image processing application execution through the least expensive and least complex hardware.

The objectives of this dissertation are: (1) to introduce a new shared-bus shared-memory multiprocessor system architecture that can maximize throughput, minimize cost and has the potential of supporting effectively real-time high-resolution image-processing; (2) to contact an extensive investigation of the performance of the proposed architecture; (3) to propose appropriate values for several system design parameters.

Indeed the proposed, in this dissertation, shared-bus shared-memory multiprocessor system can effectively support 32 DSPs. This is in sharp contrast to existing DSP-based bus-interconnected systems, which can support only a very small number of DSPs. This is achieved through the elimination of the traditional direct linkage

of the shared bus and processor data bus, which enables the utilization of a wider shared bus. Moreover, the fast prefetching cache memories and the wider shared bus provide additional bus bandwidth that can eliminate large memory latencies and improve significantly the number of effectively supported processors. It should be noted that the proposed system can maintain peak performance regardless of image size (small or large).

1.4 Outline

The remaining of this dissertation is organized as follows. Chapter 2 provides a description of the proposed twin-prefetching DSP-based shared-memory system, including a overview of the ADSP-21060. Chapter 3 introduces the methodology for the performance analysis and evaluation of the proposed system. Chapters 4 and 5 present performance results that provide a useful insight into the behavior and efficiency of the system. Chapter 6 discusses optimal size selection for prefetching caches. Finally, Chapter 7 presents the conclusions and future research.

CHAPTER 2

TWIN-PREFETCHING DSP-BASED SHARED-MEMORY SYSTEM

The proposed multiprocessor is a high-speed low-cost DSP-based *twin-prefetching* shared-memory MIMD parallel system (Figure 2.1). It consists of P nodes where P is power of two. The system is investigated for several values of P such as 1, 2, 4, 8, 16, 32 and 64. At the heart of each node, shown in Figure 2.2, is a DSP processor (ADSP-21060) optimized for image processing, graphics, speech, sound and other high-speed numeric processing applications. Several characteristics of this high-performance DSP processor (ADSP-21060) are presented in Section 2.1. Each node is also comprised of two high-speed memories with their controllers, called *Twin1* and *Twin2*. Their *twin-prefetching* operation is described in Section 2.2.

2.1 The ADSP-21060

The ADSP-21060 is a super harvard architecture processor (SHARC) [15]. It is the most versatile and powerful processor offered by Analog Devices, an industry leader in DSP technology [29]. It has four independent buses for dual data, instructions, and I/O. With its separate program and data memory buses and on-chip instruction cache, the ADSP-21060 can fetch two operands and an instruction (from the cache), all in a single cycle. The ADSP-21060 key features are:

- Single-cycle multiply & ALU operations with dual memory read/writes and instruction fetch.

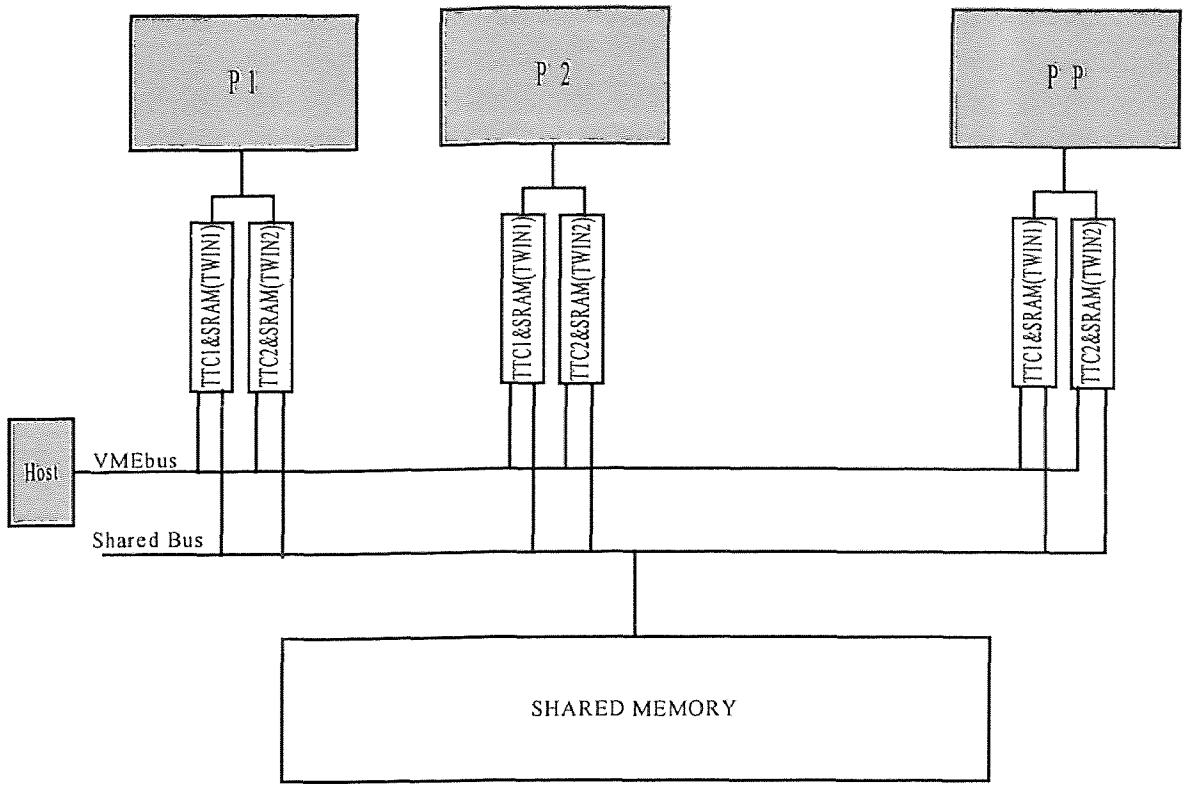

Figure 2.1 Twin prefetching multiprocessor system diagram

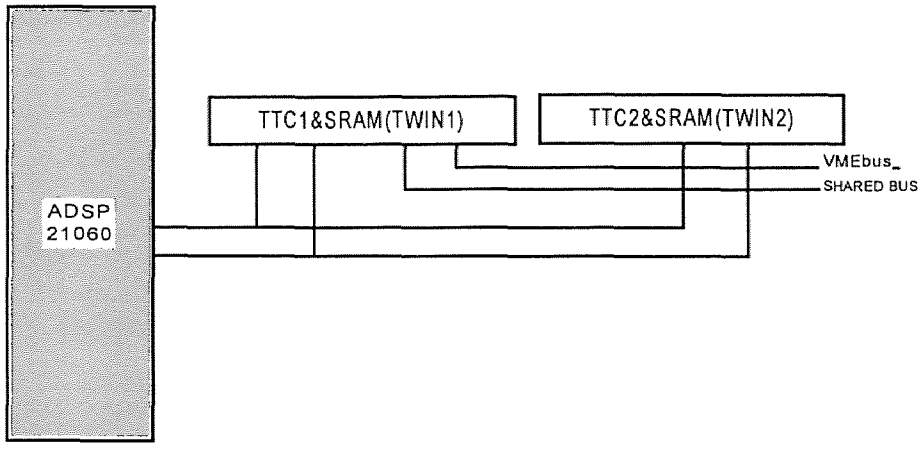

Figure 2.2 Node block diagram

- Super harvard architecture - four independent buses for dual data, instructions, and I/O
- Dual ported, for independent access by core processor and DMA, on-chip 4 Mbit SRAM and integrated I/O peripherals
- Integrated multiprocessing features
    - Glueless connection for scalable DSP multiprocessing architecture
    - Distributed on-chip bus arbitration
    - Six link ports for point to point connectivity and array multiprocessing
    - 240 Mbytes/s transfer rate over parallel bus
    - 240 Mbytes/s transfer rate over link ports
- 120 MFLOPS peak performance
- 40 MIPS, 25 ns instruction rate
- Dual data address generators with modulo and bit reverse addressing
- Efficient program sequencing with zero-overhead looping: single cycle loop setup
- 32-bit single precision & 40-bit extended-precision IEEE floating-point data formats
- 32-bit fixed-point data format, integer & fractional, with 80-bit accumulators
- 10 DMA channels
    - Background DMA transfers at 40 MHz, in parallel with full-speed processor execution

- Performs transfers between ADSP-21060 internal memory and external memory, external peripherals, host processor, serial ports, or link ports

- Three independent computational units
  - Arithmetic logic unit (ALU)
  - Multiplier/Accumulator (MAC)
  - Shifter
- Internal instruction cache
- Provisions for multiprecision computation and saturation logic
- Multifunction instructions

The ADSP-21060 flexible architecture comprehensive instruction set supports a high degree of parallelism. In one cycle the ADSP-21060 can:

- generate the next program address
- fetch the next instruction
- perform one or two data moves
- update one or two data pointers
- perform a computational operation

2.1.1 Multiprocessing

The ADSP-21060 offers powerful features tailored to multiprocessing DSP systems. The unified address space allows direct interprocessor accesses of each ADSP-21060's internal memory. Bus arbitration logic is included on-chip for simple, glueless connection of systems containing several ADSP-21060s and a host processor. Master processor (bus master) changeover incurs only in one cycle overhead. Bus arbitration is selectable as either fixed or rotating priority.

Due to the fact that gluless connection is limited only to 6 processors and the proposed system is investigated for up to 64 processors, an additional arbitration unit on the shared bus is assumed implementing rotating priority algorithm.

2.1.2 Dual-Ported Internal Memory

The ADSP-21060 contains 4 megabits of on-chip SRAM, organized as two blocks of 2 Mbits each, which can be configured for different combinations of code and data. Each memory block is dual-ported for single-cycle, independent accesses by the core processor and I/O processor or DMA controller. Memory can be accessed as 16-bit, 32-bit, or 48-bit words.

While each memory block can store combinations of code and data, accesses are more efficient when one block stores data, using the data memory (DM) bus, and the other block stores instructions and data, using the program memory (PM) bus. Thus, a dedicated bus to each memory block assures single-cycle instruction execution with two data transfers. It should be noted that dual data transfer is possible if the instruction is available in the instruction cache.

Single-cycle instruction execution is also maintained when one of the data operands is transferred to or from off-chip, via the ADSP-21060's external port. This is how the proposed *twin-prefetching* system operates, storing code and some data (filter coefficients, for example) in the internal memory and retrieving all image data from the DM data bus through the external port.

2.2 Data Memory

In programming systems, processors are generally considered active resources and memory is viewed as passive resource. For the proposed system, data memory (prefetching cache) functions as both a passive and an active resource; passive because it supplies the core processor with data and active because it initiates and completes data transfers from/to the global (shared) memory.

Data memory is comprised of two controllers (*twin TTCs)* and two fast memories (*twin-prefetching* caches) placed between the DSP processor and the shared bus interconnect. The two *TTC*/cache pairs are referred to as *Twin1* and *Twin2*. In a typical operation, one *Twin* is accessible to the processor providing data operands while the other *Twin* is transferring data from/to the shared memory.

The elimination of the traditional direct linkage of the shared and processor data bus enables the utilization of a wider shared bus, i.e., shared bus width becomes independent of the data bus width of the processors. The fast *twin-prefetching* memories and the wider shared bus provide additional bus bandwidth into the system, which eliminates large memory latencies; such memory latencies constitute the major drawback for the performance of shared-memory multiprocessors.

2.2.1 The TTCs

The *TTC1* and *TTC2* controllers are, more specifically, DMA-like devices capable of two-dimensional addressing. They move rectangular regions of data between global memory and one of the node's caches. In addition to the DMA capability, the two *TTCs* comprise a bi-directional communications port, which allows command, status, and

parameter passing between the VME-bus-connected host and the ADSP-21060. (Communication capability of *Twins* is not necessary since ADSP-21060 could connect directly to VME-bus through its own communication channels; it is required though, if we are to place a less versatile processor at the node's core.)

The *TTC* is accessed and operated through a set of registers. This group of registers appears in the ADSP-21060 data memory space. To transfer data between global memory and one of the node's prefetching caches, the ADSP-21060:

- loads the *TTC* registers with the Start and End addresses of the block of data in the global memory
- loads the *TTC* registers with the Start and End addresses of the block of data in cache
- loads the TTC CONTROL register
- sets the "start_transfer" bit in the CONTROL register and the transfer begins

The direction of the transfer depends on the value of a specific bit in the control register. Loading and unloading the *Twins* occur simultaneously with data processing. In other words, as soon as a block of data has been moved into the cache, the ADSP-21060 begins processing it, while the other cache is emptied and then filled with new data. The processor finishes processing data in *Twin1*, and switches to *Twin2*, which is filled with fresh data. The back and forth switching of *Twin1* and *Twin2* allows maximum utilization of resources, and thus optimum system performance.

For P processing elements a maximum of 2P *TTCs* compete for the bus which is granted to the *TTC* by an arbitrator implementing rotating priority. Processor P_i is

serviced before processor P_{i+1} and *Twin1* is serviced before *Twin2*. More precisely, if $Twin_{ij}$ is the j^{th} *Twin* of the i^{th} processor, the rotation proceeds as follows: $Twin_{11}$, $Twin_{21}$,..., $Twin_{p1}$, $Twin_{12}$, $Twin_{22}$,..., $Twin_{p2}$.

2.3 Host Processor

Host processor is responsible for booting all nodes and downloading all necessary code and some data to the internal memory of every ADSP-21060. The data downloaded to the internal memories include the addresses of image segments in the global memory which every node is assigned to process. These addresses are the result of partitioning (a technique for decomposing a large data set into many small pieces for parallel execution by multiple processors). These addresses are calculated once for a specific image size and application and are available for all future requests.

Asynchronous transfers at speeds up to the full clock rate of the processor (ADSP-21060) are supported. The host interface is accessed through the ADSP-21060's external port and is memory mapped into the unified address space. Four channels of DMA are available for the host interface. The host processor requests the ADSP-21060's external bus with the host bus request (HBR), host bus grant (HBG) and ready (REDY) signals. The host can directly read or write the internal memory of the ADSP-21060, and can access the DMA channel set up and mailbox registers.

2.4 Theoretical Analysis

The performance of a computer system, in general, is greatly affected by the amount of locality present in code or data of an application. Traditional caches and prefetching

caches were invented to take advantage of the temporal, spatial, and sequential locality present in applications. Spatial locality is the tendency of a process to access items whose addresses are near one another i.e., indicates that if a location in memory is accessed, then others nearby will probably be accessed soon. Temporal locality denotes that if an address in memory is accessed once, then it will probably be accessed again soon. Lastly, sequential locality indicates the sequential order of most instructions in code.

DSP algorithms are usually encoded in relatively small programs, which occupy moderate amounts of program memory. Our system could easily store them in the internal on-chip memory allowing single cycle multiple instruction execution if image data can be retrieved as fast as well. The real challenge is data memory. It becomes prohibitively costly to store several large high-resolution images on expensive fast data memories (SRAM). *Twin* prefetching of image segments (along with the use of a wider shared bus) is a cost effective solution for literally taking advantage of every clock cycle of every DSP, i.e., we obtain 100% cache hit ratio (if there is enough shared-bus bandwidth not to stall the prefetching mechanism).

2.4.1 Partitioning Images into Cache Prefetching Segments

Images are stored in the shared memory (inexpensive DRAM). All data are partitioned into several segments. The segments' addresses could be calculated either by the host or node processor. For our system the host processor calculates and downloads the addresses to the nodes. Once calculated, the addresses are stored in the local memory of the host processor and are available when needed again. A segment should be smaller than the prefetching cache size for two reasons:

- For the segment to fit in the cache.
- To leave some space for the produced output.

We should note that even if a quite large cache prefetching memory is available, a smaller segment may eventuate a faster system execution time. In other words, there should be enough prefetching cache memory to satisfy the most demanding applications (prefetching cache size demanding applications). It is not necessary to utilize all of it.

If P processing elements are available, the image is divided into *l*xP segments, i.e., *l* segments of data are allocated to every node. (Adjacent segments may overlap each other in certain applications). Interchangeably *TTC1* and *TTC2* are prefetching the segments in their respective caches. The node processor switches back and forth processing data in *Twin1* and *Twin2*. Constantly retrieving data from fast prefetching caches, processors maintain single cycle instruction execution time. Continuous data retrieval from prefetching caches eliminates shared bus latencies and maximizes system performance. Data segments should weight equally, (contain the same amount data-take the same amount of processing time) as much as possible, in order for the system to be well balanced-a basic condition for obtaining maximum throughput.

2.4.1.1 Example - Two Dimensional Convolution Partitioning Rule: Let P be the number of processing elements P_1, P_2,..., P_i,...,P_p in the system. Let the input be an M_rxN_c image where M_r is the number of pixel rows numbered 0,1,...,M_r-1 while N_c is the number of pixel columns numbered 0,1,...,N_c-1. Let also a template matrix m_rxn_c where m_r is the number of rows and n_c is the number of columns of the matrix.

If P processors take part in the convolution, the image is partitioned into P sections. One way to do that is to divide M_r by P. Assuming M_r is divisible by P, all sections consist of the same number of rows. For the convolution algorithm $k+m_r-1$ input rows are required to produce k output rows. Taking this into consideration every section should include an additional m_r-1 rows from the following section below. The last section at the bottom of the image wraps around (to the first section). The total number of rows in a section becomes $M_r/P+m_r-1$.

Partitioning assigns the i_{th} section to processing element P_i. We define r_i the range of rows in the i_{th} section. r_i is then described by

$$(i-1)\frac{M_r}{P} \le r_i \le (i)\frac{M_r}{P} + (m_r - 1) - 1 \qquad (2.1)$$

Every section consists of l segments (smaller divisions of a section which are prefetched interchangeably by the *Twins*). If y output rows are produced for every prefetched segment, $l = M_r/P/y$ (l is usually proportional to prefetching input and not output, but it so happens that in the two-dimensional convolution algorithm the number of prefetching input rows is equal to $y+m_r-1$).

Let *begin_row_pref_addr* and *end_row_pref_addr* be the two row addresses that include all data in a segment to be prefetched. All addresses of segments in the global memory are readily determined according to the following simple code:

begin_row_pref_addr = (initial value)
end_row_pref_addr = begin_row_pref_addr + y + m_r – 2
loop
begin_row_pref_addr = begin_row_pref_addr + y

end_row_pref_addr = begin_row_pref_addr + y + m_r – 2
jump to loop

Column addresses remain constant for all prefetching segments. They receive their values from the start and end column address of the image in the global memory.

Let us give some specific values to the variables in Equation 2.1. Let M_r=1024. If P = 2, then i = 1,2. Substituting i with 1 and then 2 in the equation we find $0 \le r_1 \le 511+(m_r-1)$ and $512 \le r_2 \le (1023+(m_r-1$ (wrap around))). Ranges r_1 and r_2 include all segments assigned to P_1 and P_2 respectively. If y = 4 then l = 128 and every segments consists of 4 + (m_r-1) pixel rows. Likewise, if P=4, then i= 1,2,3,4, and $0 \le r_1 \le 255+(m_r-1)$, $256 \le r_2 \le 511+(m_r-1)$, $512 \le r_3 \le 767+(m_r-1)$, and $768 \le r_4 \le (1023+(m_r-1$ (wrap around))). If y = 4 then l = 64 and every segments consists of 4 + (m_r-1) pixel rows.

2.4.2 Categories of Applications

From this point on we will refer to t_W as the time required by the processor to process a segment of data, t_{UL} as the sum of t_L (time required by a *TTC* to load a segment in the prefetching cache) and t_U (time required to unload the results produced from processing the previously loaded segment), and *Twratio* as the ratio t_W/t_{UL}. It should be noted that in this section we assume that arbitration delays and processor switching from one *Twin* to the other are considered negligible. Negligible are also considered the few clock cycles which the processor spends to transfer to a *Twin* the address of the next segment. Simulation, though, takes into consideration all small details of the multiprocessor system.

Different applications require different t_W to process the same amount of data. The larger the t_W the more bus bandwidth available to serve a greater number of nodes. (During time t_W the *Twin* does not hold or request the shared bus).The *Twratio* (t_W/t_{UL}) is extremely important for the *twin-prefetching* multiprocessor system. If *Twratio*'s value is equivalent to P (Eq. 2.2), the *twin-prefetching* multiprocessor system reaches maximum utilization of its resources (i.e., bus and processors bandwidth). *Twratio*s greater than P move the system further away from bus contention. *Twratio*s less than P, on the other hand, induce bottleneck and limit multiprocessor's throughput.

$$t_W/t_{UL} = P \tag{2.2}$$

The boundary condition (t_W/t_{UL}=P), serves as a basis in order to comprehend the operation and capabilities of the *twin-prefetching* multiprocessor system. It could be proven simply as follows: For P processing elements there are 2P *Twins* in the system sharing the common bus with rotating priority. Let T_{rot} be the total time for all *Twins* to unload results and load a new segment of image data. By definition, at the boundary condition there is just enough bus bandwidth for all *Twins* to have their segments ready for processing by their core processors. Therefore T_{rot} is given by

$$T_{rot} = 2 \times P \times t_{UL} \tag{2.3}$$

During time T_{rot} a processor has just enough time to process data in both *Twins*, i.e., finishes processing *Twin1* and immediately switches to *Twin2* without having to wait for any data transfer to finish. Therefore T_{rot} is also given by

$$T_{rot} = 2 \times t_W \qquad (2.4)$$

From Equations 2.3 and 2.4 we derive the boundary Equation 2.2.

Hence every application falls into one of three possible cases:

1. $(t_W/t_{UL}) = P$ maximum utilization of bus and processor bandwidth
2. $(t_W/t_{UL}) > P$ unused shared bus bandwidth available
3. $(t_W/t_{UL}) < P$ bottleneck- unused processor bandwidth

Cases one and two are preferable since bottleneck is undesirable in all multiprocessor systems.

The *twin-prefetching* multiprocessor system is expected to perform well over other existing shared memory multiprocessor systems because the "*twin*" architecture allows *cache-only* data retrieval. An invaluable advantage over other systems is the elimination of the traditional direct linkage of the shared bus and processor data bus which makes feasible the utilization of a wider shared bus.

2.5 Simulation

Simulation model consists of the ADSP-21060, two controllers (twin TTCs), two fast memory modules, a shared memory and a shared-bus interconnect. The most difficult part of the simulation was the decision about the location of timers. Usually timing units are the processing units themselves. In our system the processing units do not communicate directly with the shared-bus; they communicate through TTCs. Therefore, timers were

placed on TTCs which monitor both the shared-bus and processor activity. If there is adequate bandwidth on the bus (for image segments to be loaded on time and to be ready in prefetching caches for processing) then timer of TTC1 starts where the timer of TTC2 stopped and vise versa. If not sufficient bandwidth is available, then delays are encountered on the shared bus which have to be considered. Accessing the cache takes 25 *ns* (the same as the instruction rate) while accessing shared-memory takes 75 *ns*. Arbitration costs 4 clock cycles, processor switching from one twin to the other costs 1 clock cycle, and informing the TTC about the next unload and load costs 20 cycles. All instructions are executed in one clock cycle. Multifuntion instructions with dual data fetches are executed in one clock cycle only if they are available in instruction cache. Hence all multifuction instructions which implement the convolution algorithm (appendix A) execute in two clock cycles the first time the loop is entered; in any subsequent loop they are executed in one clock cycle.

CHAPTER 3

PERFORMANCE ANALYSIS

The DSP-based *twin-prefetching* shared-memory multiprocessor is investigated for P processing elements, several shared-bus interconnect widths (*nl*) and several prefetching cache sizes (*csz*). P receives values such as 1, 2, 4, 8, 16, 32 and 64 and *nl* receives values such as 32, 64, 128, 256 and 512 while *csz* receives 10 different values between a few *Kbytes* and 4 *Mbytes*. Every possible combination of P, *nl*, and *csz* define a multiprocessor system with different capabilities. All system configurations are numbered 350. We specifically investigate the following:

- which configurations of *twin prefetching* multiprocessor system perform better
- how many nodes could be effectively supported by the shared-bus interconnect
- the optimal prefetching cache size
- the effect of changing the shared-bus-width (*nl*) on the performance of *twin-prefetching* multiprocessor system
- the effect of the wide variance of temporal and spatial locality (present in different image processing applications) on the performance of *twin-prefetching* multiprocessor systems
- the effect of the wide variance of temporal and spatial locality of applications on the prefetching cache size

Timing the execution of several image-processing applications would have been a straightforward but inaccurate way to investigate all configurations of *twin-prefetching* system. One reason is that the quantity of spatial and temporal localities (factors that significantly affect *Twratio*) are not readily observed in different image-processing applications; thus results would be quite deceiving if applications not having a wide range of *Twratios* were selected. Furthermore, if the selection of some applications for the investigation of one system is usually a debating subject for researchers one could imagine the degree of disagreement when a number of systems is involved. Another reason is that different applications may require different partitioning algorithms, which possibly divide the image into significantly dissimilar segments. Thus making unfeasible the fair performance comparison of systems. It would be impossible, for example, to determine whether the source of a resulted significant change in the value of the optimal prefetching cache size is the application itself or some unanticipated (before simulation) operational detail affecting *twin-prefetching*.

3.1 Methodology for Performance Evaluation

In order to overcome the above problems a fair methodology is devised in order to investigate and evaluate the performance of the 350 different configurations of *twin-prefetching* shared-memory system when executing computationally intensive high-resolution image-processing applications. This methodology is based on finding first the main characteristics of image-processing applications when implemented on the *twin-prefetching* system. Afterwards we apply low and high end values of the characteristics aiming to cover all possible cases.

The factor, which includes all characteristics of an application executed on the proposed system, is *Twratio* (t_W/t_{UL}). According to the value of *Twratio,* a particular application falls into one of three cases named Case I, Case II, and Case III:

I. *Twratio* = P maximum utilization of bus and processor bandwidth

II. *Twratio* > P unused shared-bus bandwidth available

III. *Twratio* < P bottleneck - unused processor bandwidth

All cases are also discussed in Chapter 2 where it is established that *Twratio* is the determining factor for the performance of an application on any configuration of the *twin-prefetching* multiprocessor. It inherently combines all the fundamental software and hardware parameters. Some of the parameters are: amount of spatial and temporal locality involved with the application (a major contributor to t_W), speed of DSPs, access time of both shared-memory and prefetching cache memories, partitioning algorithm of input images (prefetching segment size), produced output segment size, shared-bus-interconnect-width, communication overhead, P, and the shared-bus bandwidth. If hardware parameters are set to specific values, then *Twratio* depends only on application parameters like:

- t_W
- input segment size & t_L

- produced output size & t_U
- communication overhead

(It should be noted that the above parameters are both hardware and software dependent. They are referred as application dependent when hardware parameters receive constant values.)

t_W (the time required by the processor to process a segment of data) is proportional to spatial and temporal locality of the application, i.e., the more times memory-addresses, of data in the prefetched segment, are accessed the larger the value of t_W. In other words, t_W is analogous to the (average) number of times which every item in the prefetched segment is referenced. t_L is dependent on the size of input segment. If the segment size is increased, t_L and t_W are also increased. t_U is also generally increased. (In very rare cases there is very little output, which is transferred to the shared memory at the last unload.) Regardless though, of how much every parameter value increases or decreases what really matters is *Twratio* (t_W/t_{UL}).

We investigate the effect of different t_Ls and t_Us by varying the prefetching segment size. Moreover, varying the size of the input segment in the prefetching cache (taking into account the change of output size too), we evaluate, also, the performance of different sizes of prefetching cache. It should be noted that there are obvious and not so apparent factors determining the optimal size of prefetching cache. Obvious and predictable factors are: the size of input segment and the size of produced output. The less obvious factors, which might affect the optimal prefetching cache size and the operation

of the system in general, are the various delays due to initial loading of segments (transient performance), amount of time spent by the processor to instruct the *Twins* about the next loading and unloading, amount of time spent by the processor to switch from one *Twin* to the other, and possible unknown effects of the *twin-prefetching* mechanism.

Finally, communication overhead (h) should be a small fraction of the workload (w) in a parallel algorithm in order for the algorithm to be efficiently mapped onto a parallel computer. The smaller the communication overhead the closer the efficiency (E) to 1. Generally the efficiency of a parallel algorithm is defined as

$$E = \frac{w}{w + h} \tag{3.1}$$

We intent to show that the *Twin* prefetching system could handle great amount of communication operations with little overhead due to the *twin-prefetching* mechanism and additional bandwidth provided by a wider shared-bus.

We could summarize what was said so far in this chapter with two contradictory statements.

1. We need the characteristics of several applications to investigate and evaluate all 350 hardware configurations of the *twin-prefetching* system.
2. Timings several different applications would not support a fair comparison of all *twin-prefetching* systems.

All the above challenges are met by selecting an application, which behaves like many applications if its attributes are judiciously varied.

3.2 Two-Dimensional Convolution

The selected application is two-dimensional convolution, which serves as the fundamental operation for a wide variety of other image-processing and computer vision applications, e.g., edge detection, object detection, image smoothing [16], edge enhancement. Because of the fundamental nature of this application problem and because of its high computing complexity on a single processor system, much attention has been devoted to the development of efficient parallel architectures and algorithms for its implementation [38][39][40][41]. The inputs to the two-dimensional convolution are an M_rxN_c image matrix I[0...(M_r-1)] [0...(N_c-1)] and an mxn template matrix T[0...(m_r-1)] [0...(n_c-1)]. The output image is an M_rxN_c matrix C_{2D} where:

$$C_{2D}[i,j] = \sum_{u=0}^{m_r-1} \sum_{v=0}^{n_c-1} I[(i+u),(j+v)] \bullet T[u,v] \tag{3.2}$$

for $0 \le i \le M_r-1$ and $0 \le j \le N_c-1$.

C_{2D} is called the two-dimensional convolution of I and T. When implementing two-dimensional convolution on the *twin-prefetching* multiprocessor, template matrix is allowed to extend outside the input window (wraparound); otherwise, the output matrix is smaller than the input matrix by m_r-1 rows and n_c-1 columns.

Input image matrix I is partitioned among processors according to the rule described in section 2.4.1.1 assuming that M_r and N_c are powers of 2.

Let us examine, in more detail, how convolution serves as the foundation for the fair investigation and evaluation of all 350 hardware configurations of *twin-prefetching* system, i.e., how parameters of the convolution problem receive values, such that they cover a wide variety of applications.

3.3 Application Parameters

Different *Twratios* ($t_W/(t_L+t_U)$) are produced by varying the size of the template matrix. By increasing the template size we increase t_W (processing time of a segment) while keeping the size of input segment (and consequently t_L) virtually the same. Spatial and temporal locality of the application (we refer to locality of data of the application, not code) is also increased. Measure for locality is the number of times which every segment pixel is referenced in prefetching caches. Typically every datum in caches or prefetching caches is referenced from a few to about 30 times. Template matrix sizes of 2x2, 3x3, 6x6, and 9x9 are utilized, yielding localities of 4, 9, 36, and 81, respectively, thus, covering a wide range of data localities and *Twratios*.

Investigation of the effect of prefetching cache size on performance is accomplished by keeping all other hardware and software parameters constant and varying the size of input segment in the prefetching cache from a few *Kbytes* to 4 *Mbytes*. Ten different sizes of input image segment cover a wide range of t_L, t_U, and prefetching cache size.

Before explaining the way we introduce communication overhead to the system,

let us be reminded that multicomputers (distributed memory systems) communicate through message-passing among nodes while multiprocessors (shared memory systems) communicate with each other through the shared-memory. In the case of convolution, if α output matrix rows are to be produced $\alpha+m_r-1$ rows have to be present. Let us assume that the image is partitioned dividing the number of rows by the number of processing elements. In a multicomputer system every processor would have to transfer (communicate) the m_r-1 rows from other nodes in order to produce the α output rows. In a bus connected shared-memory system every node retrieve extra m_r-1 rows (besides the equal shared from dividing all rows by P) in order to produce the α output rows. In general, when solving convolution using a parallel computer and not a uniprocessor system $P(m_r-1)$ extra rows of data have to be communicated. The ratio $(m_r-1)/\alpha$ is a direct measure of the communication overhead involved and its value is traditionally expected to be small (much smaller than 1).

We test the *twin-prefetching* shared-memory system under severe communication overhead by allowing the ratio $(m_r-1)/\alpha$ to take values much larger than one. For example if $\alpha=1$ (in a prefetched segment) and template matrixes 2x2, 3x3, 6x6, 9x9 are used, the value of m_r-1 becomes 1,2,5,8 respectively. The *twin-prefetching* shared memory system is also tested under lighter communication overhead. We introduce extreme ratios like 1/1, 2/1, 5/1, and 8/1, in order to investigate the performance of the proposed architecture with applications employing large amount of communication overhead. It is important to note that bus connected shared memory systems are extremely sensitive to the amount of communication overhead. This is due to the fact that all system activity should "fit in

only one path." For this reason DSP-based bus connected architectures support only up to 4 processors. We expect the proposed architecture to effectively support more than four due to the *twin-prefetching* mechanism and wider shared-bus.

The convolution is performed on eight continuous image frames stored in the shared memory. The resolution of every image is 1024x1024. We choose to test the proposed architecture with large, high-resolution images because we want to show that this architecture can sustain maximum performance in challenging processing conditions. Existing systems claim real-time execution of small (64x64 or 128x128) images, which fit in small local or internal SRAM memories. If they are to perform operations on larger images their performance degrades dramatically. The *twin-prefetching* architecture does not have these limitations.

The execution time of an application on the proposed system is the time after all output results are transferred in the shared memory. This is in contrast to some investigations of distributed or shared-memory systems, which present impressive but deceiving results by measuring execution time excluding time to load and time to unload results to/from local memories.

CHAPTER 4

EFFECT OF SHARED-BUS-WIDTH ON PERFORMANCE

Chapter 4 investigates the effect of shared-bus-interconnect-width on the performance of several configurations of the proposed P-node DSP-based *twin-prefetching* shared-memory multiprocessor systems. P receives values 1, 2, 4, 8, 16, 32 and 64 and shared-bus-width (*nl*) receives values 32, 64, 128, 256, and 512. Tables and figures in this chapter demonstrate how the performance of a P-node *twin-prefetching* multiprocessor system changes as shared-bus-width changes. The number of nodes (processors) which can be effectively supported for a specific value of shared-bus-width is also investigated. Prefetching cache size value was kept between 262 *Kbytes* and 290 *Kbytes*. The selection of prefetching cache size is analyzed in Chapter 6. Not too much attention should be paid to the specific prefetching cache size since in most cases prefetching cache sizes ranging from ~100 *Kbytes* to ~300 *Kbytes* yield performances with no more than 2% difference.

Four applications (convolution utilizing *four* different *template matrices*) were executed for all multiprocessor system configurations. Timings (T) are based on convolution of *eight*, continuous high-resolution images (1024x1024), stored in shared memory. Pixel size is 16 bits.

Every application demands different amount of shared-bus bandwidth. In Sections 4.1, 4.2, 4.3, and 4.4 we investigate the performance of several P-node systems for several values of *nl* when the template matrix is 2x2, 3x3, 6x6, and 9x9 respectively. The reason for focusing at the timings of individual applications (one by one) is to ensure that

the observed changes in availability of shared-bus bandwidth is due *only* to the change of the shared-bus width, not on the application. With the same criteria the fair comparison of all configurations of *twin-prefetching* multiprocessor system is also ensured. Finally, a small discussion on all applications is provided in Section 4.5.

Speedup factor S(P), indicates how much faster a P-node system executes an application compared to a uniprocessor system (*nl* is kept the same in both cases). *S(P)* is given by

$$S(P) = \frac{T(1)}{T(P)} \tag{4.1}$$

where *T(1)* is the time required to execute a program on a uniprocessor while *T(P)* is the time required to execute the same program on a P-node (processor) system. We define the *bus-width-speedup-factor S(nl)*, as the ratio of *T(32)* and *T(nl). T(32)* is the time required to execute a program on a P-node system with shared-bus width equal to 32, and *T(nl)* is the time required to execute a program on the same P-node system with shared-bus width equal to *nl*. Thus *S(nl)* is given by

$$S(nl) = \frac{T(32)}{T(nl)} \tag{4.2}$$

System efficiency, E(P), of a P-node multiprocessor is given by

$$E(P) = \frac{S(P)}{P} \tag{4.3}$$

The best possible *system efficiency* is 1, i.e., the best *speedup factor* is linear, or *S(P)*=P.

Tables labeled "Time vs. Number of Processors" consist of *six* columns. The first column contains P values, while the other *five* columns contain timings of the P-node system when *nl* is 32,64,128,256, and 512. The corresponding speedup, S(P), and efficiency, E(P), of every entry is given in Tables labeled "Speedup vs. Number of Processors" and "Efficiency vs. Number of Processors," respectively.

Tables labeled "Speedup vs. Shared-bus-width" consist of *eight* columns. The first column contains shared-bus-width (*nl*) values, while the other *seven* columns contain the speedup factor of a specific P-node system for every value of *nl*. SP1 in these tables stands for S(1), SP2 stands for S(2), and so on. Appendix B contains tables and figures describing "Execution Time vs. Shared-bus-width," and the corresponding tables and figures for "Speedup vs. Shared-bus-width."

An effective system is considered to be a system with a speedup factor greater than P/2, i.e., providing system efficiency greater than 0.5. Shaded areas within tables labeled "Time or speedup or efficiency vs. Number of Processors," point out effective systems.

4.1 Effect of Shared-Bus Width on Performance when Template Matrix=2x2

Tables 4.1, 4.2 and 4.3 and Figures 4.1, 4.2 and 4.3 show that Convolution with a *template matrix* of 2x2 is effectively executed by up to *four* processors if *nl*=32, by up to

four processors if *nl*=64, by up to *eight* processors if *nl*=128, by up to *16* processors if *nl*=256, by up to *32* processors if *nl*=512.

Figures 4.2 and 4.3 and Tables 4.2 and 4.3 show near perfect speedup factors and system efficiencies (E≥0.90) for *nl*=32&P=2, for *nl*=64&P=2,4, for *nl*=128&P=2,4, for *nl*=256&P=2,4,8, and for *nl*=512&P=2,4,8,16.

Table 4.4 and Figure 4.4 show remarkable speedups of specific P-node systems, which are due o*n*ly to *nl* increase. In Appendix B, Table B.1 and Figures B.1 and B.2 show, in more detail, how the execution time decreases by increasing the shared-bus-width. Observing, for example, the first (0.6393 *sec*) and the last (0.0418 *sec*) entry of the seventh column (corresponding to the 32-node system) in Table B.1, we measure a *shared-bus-speedup-factor* of 15.6. Table 4.4 and Figure 4.4 show a maximum *shared-bus-speedup-factor* of 1.7, 1.7, 2.9, 5.7, 11.2, 15.6 and 16 for 1-node, 2-node, 4-node, 8-node, 16-node, 32-node, and 64-node systems, respectively.

Table 4.1 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Time vs. Number of Processors, *template matrix*=2x2					
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.4803	1.1600	0.9999	0.9198	0.8798
2	0.7413	0.5803	0.5001	0.4600	0.4399
4	0.6393	0.3226	0.2514	0.2307	0.2203
8	0.6393	0.3197	0.1621	0.1170	0.1110
16	0.6393	0.3197	0.1599	0.0819	0.0572
32	0.6393	0.3197	0.1599	0.0800	0.0418
64	0.6393	0.3197	0.1599	0.0800	0.0400

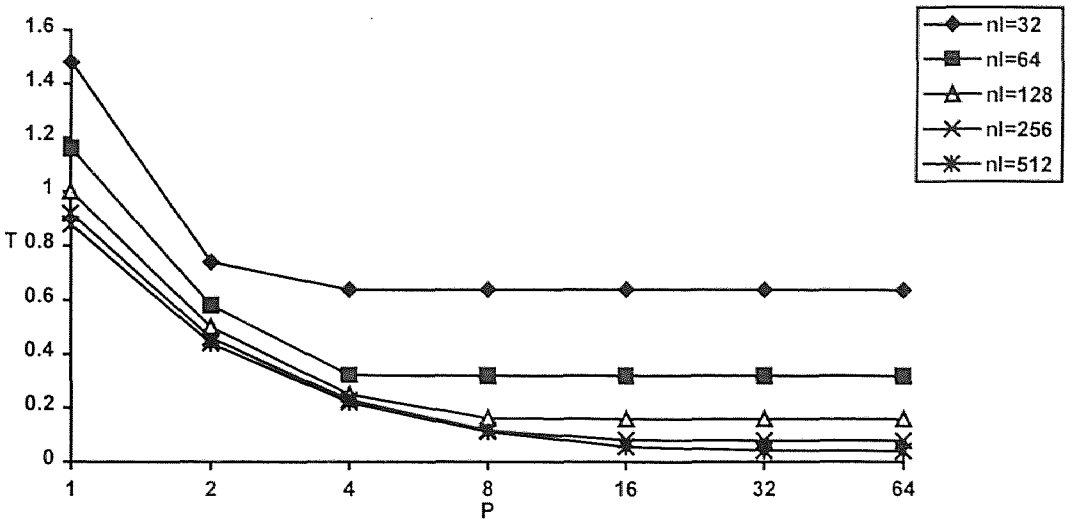

Figure 4.1 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Table 4.2 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Speedup vs. Number of Processors, *template matrix*=2x2					
	S	**S**	**S**	**S**	**S**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.9969	1.9990	1.9994	1.9996	2.0000
4	2.3155	3.5958	3.9773	3.9870	3.9936
8	2.3155	3.6284	6.1684	7.8615	7.9261
16	2.3155	3.6284	6.2533	11.2308	15.3811
32	2.3155	3.6284	6.2533	11.4975	21.0478
64	2.3155	3.6284	6.2533	11.4975	21.9950

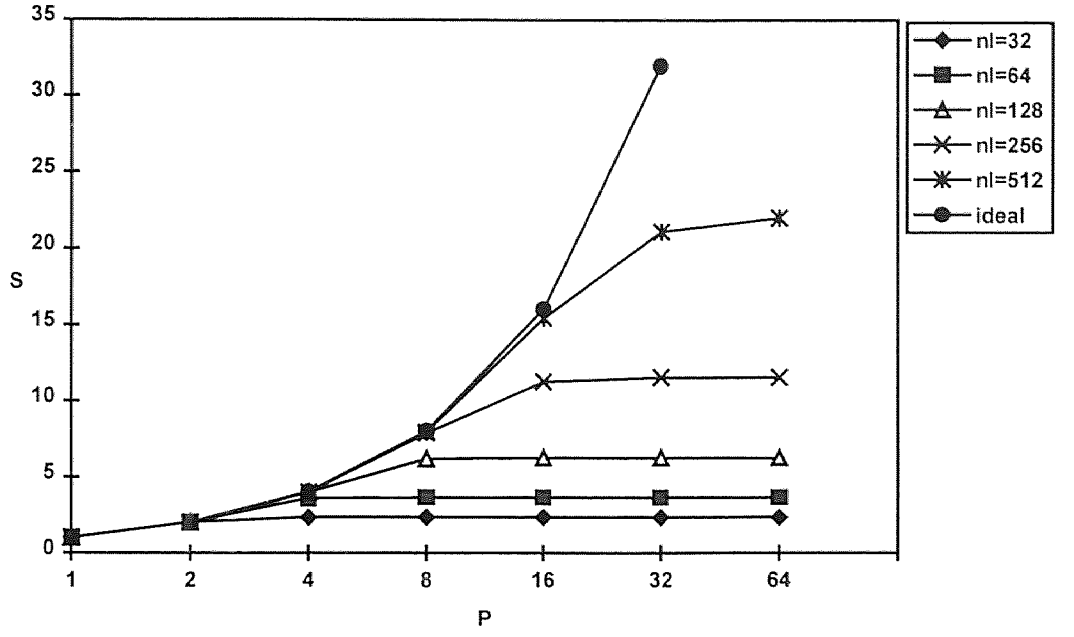

Figure 4.2 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Table 4.3 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Efficiency vs. Number of Processors, *template matrix*=2x2					
	E	**E**	**E**	**E**	**E**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9984	0.9995	0.9997	0.9998	1.0000
4	0.5789	0.8989	0.9943	0.9967	0.9984
8	0.2894	0.4536	0.7711	0.9827	0.9908
16	0.1447	0.2268	0.3908	0.7019	0.9613
32	0.0724	0.1134	0.1954	0.3593	0.6577
64	0.0362	0.0567	0.0977	0.1796	0.3437

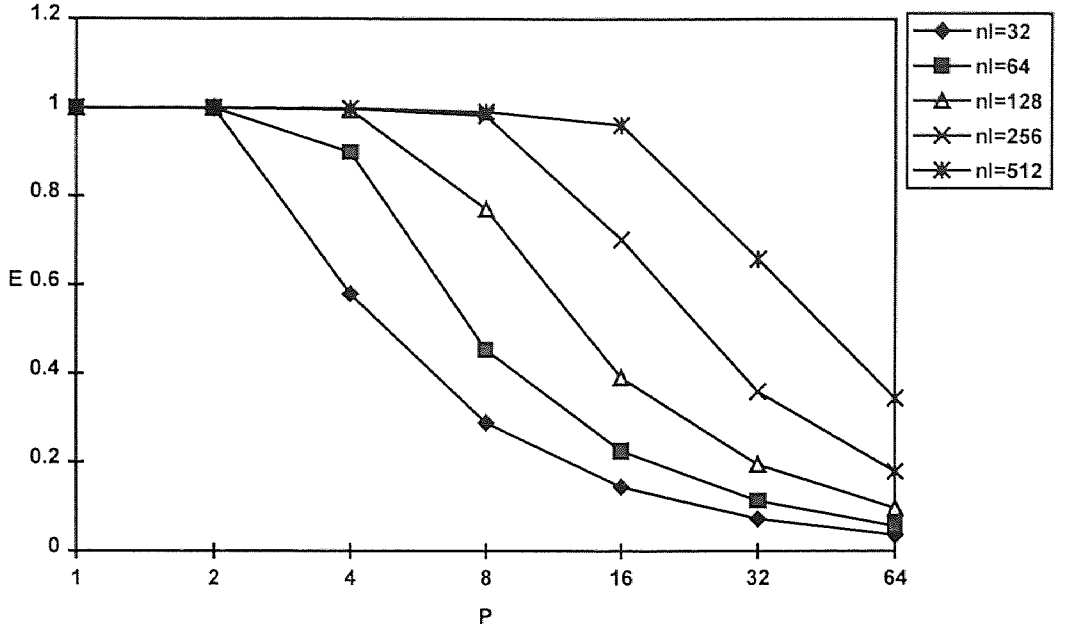

Figure 4.3 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=2x2

Table 4.4 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

Speedup vs. Shared-bus width, *template matrix*=2x2							
nl	SP1	SP2	SP4	SP8	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.2759	1.2776	1.9845	2.0031	1.9969	1.9969	1.9969
128	1.4800	1.4820	2.5458	3.9444	3.9938	3.9938	3.9938
256	1.6087	1.6109	2.7662	5.5086	7.7927	7.9875	7.9875
512	1.6818	1.6841	2.9045	5.7568	11.2105	15.5854	15.9750

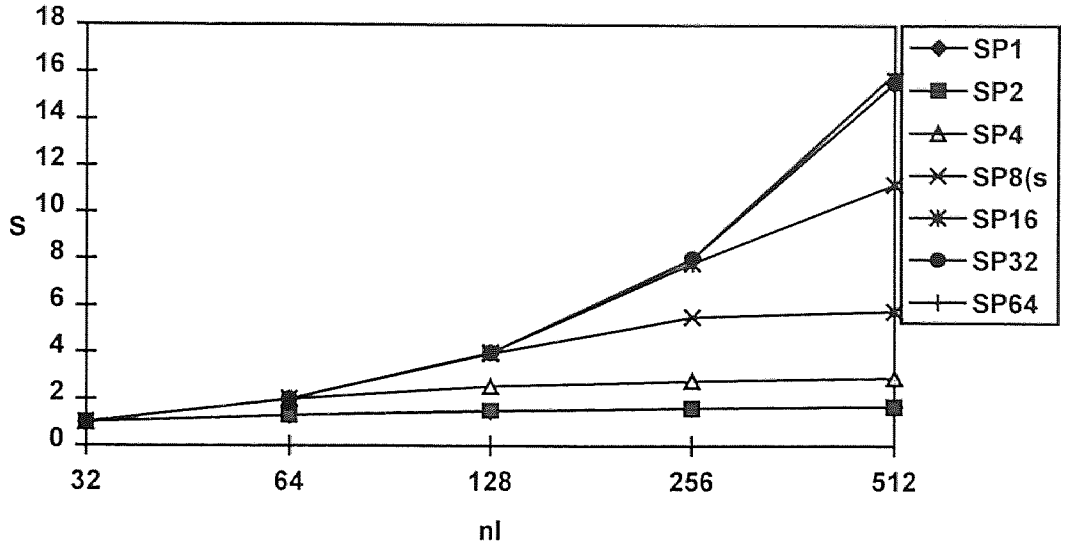

Figure 4.4 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

4.2 Effect of Shared-Bus Width on Performance when Template Matrix =3x3

Tables 4.5, 4.6 and 4.7 and Figures 4.5, 4.6 and 4.7 show that Convolution with a *template matrix* of 3x3 is effectively executed (E≥0.50) by up to *four* processors if *nl*=32, by up to *eight* processors if *nl*=64, by up to *16* processors if *nl*=128, by up to *32* processors if *nl*=256, by up to *64* processors if *nl*=512.

Figures 4.6 and 4.7 and Tables 4.6 and 4.7 show near perfect *speedup factors* and *system efficiencies* (E≥0.90) for *nl*=32&P=2,4, for *nl*=64&P=2,4, for *nl*=128&P=2,4,8, for *nl*=256&P=2,4,8,16, for *nl*=512&P=2,4,8,16,32.

Table 4.8 and Figure 4.8 show remarkable speedups of specific P-node systems, which are due *only* to *nl* increase. In Appendix B, Table B.3 and Figures B.4 and B.5

show, in more detail, how the execution time decreases by increasing the shared-bus-width. Observing, for example, the first (0.6495 *sec*) and the last (0.0457 *sec*) entry of the last column (corresponding to the 64-node system) in Table B.3, we measure a *shared-bus speedup factor* of 14.2. The value 14.2 can be confirmed by looking up the entry with coordinates *nl*=512&P=64 in Table 4.8. Table 4.8 and Figure 4.8 show a maximum *shared-bus-speedup-factor* of 1.3, 1.3, 1.4, 2.7, 5.3, 10.0 and 14.2 for 1-node, 2-node, 4-node, 8-node, 16-node, 32-node, and 64-node systems, respectively.

Table 4.5 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Time vs. Number of Processors, *template matrix*=3x3					
	nl=32	*nl*=64	*nl*=128	*nl*=256	*nl*=512
P	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
1	2.5391	2.2137	2.0510	1.9697	1.9290
2	1.2701	1.1071	1.0257	0.9849	0.9645
4	0.6569	0.5564	0.5142	0.4932	0.4826
8	0.6495	0.3309	0.2604	0.2482	0.2421
16	0.6495	0.3248	0.1679	0.1277	0.1229
32	0.6495	0.3248	0.1624	0.0864	0.0651
64	0.6495	0.3248	0.1624	0.0812	0.0457

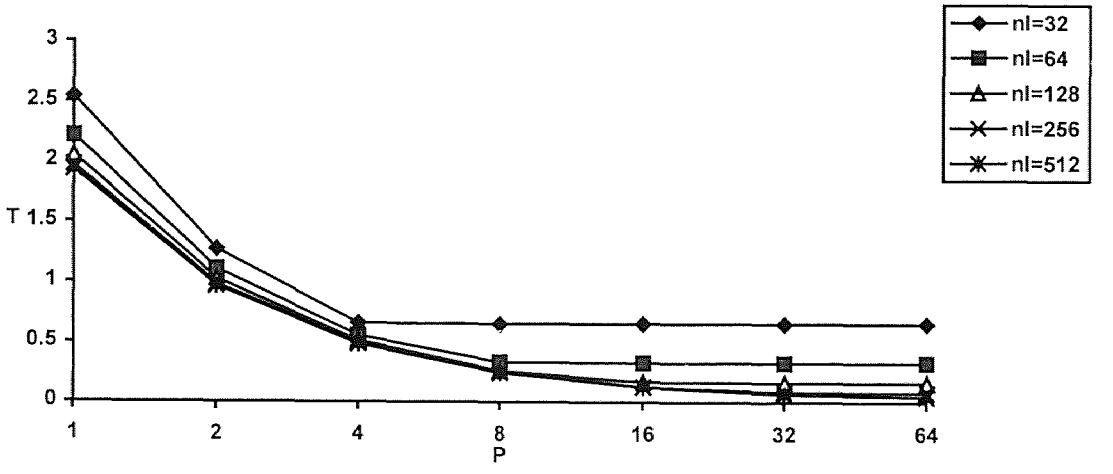

Figure 4.5 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Table 4.6 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Speedup vs. Number of Processors, *template matrix*=3x3					
	S	S	S	S	S
P	*nl*=32	*nl*=64	*nl*=128	*nl*=256	*nl*=512
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.9991	1.9995	1.9996	1.9999	2.0000
4	3.8653	3.9786	3.9887	3.9937	3.9971
8	3.9093	6.6899	7.8763	7.9359	7.9678
16	3.9093	6.8156	12.2156	15.4244	15.6957
32	3.9093	6.8156	12.6293	22.7975	29.6313
64	3.9093	6.8156	12.6293	24.2574	42.2101

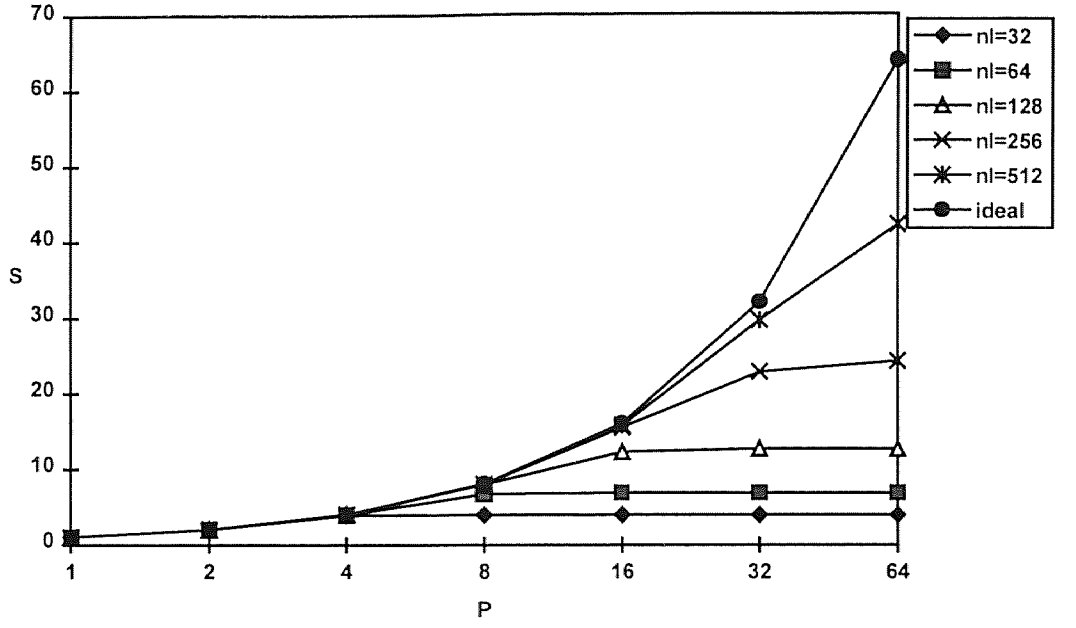

Figure 4.6 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Table 4.7 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Efficiency vs. Number of Processors, *template matrix*=3X3					
	E	**E**	**E**	**E**	**E**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9996	0.9998	0.9998	0.9999	1.0000
4	0.9663	0.9947	0.9972	0.9984	0.9993
8	0.4887	0.8362	0.9845	0.9920	0.9960
16	0.2443	0.4260	0.7635	0.9640	0.9810
32	0.1222	0.2130	0.3947	0.7124	0.9260
64	0.0611	0.1065	0.1973	0.3790	0.6595

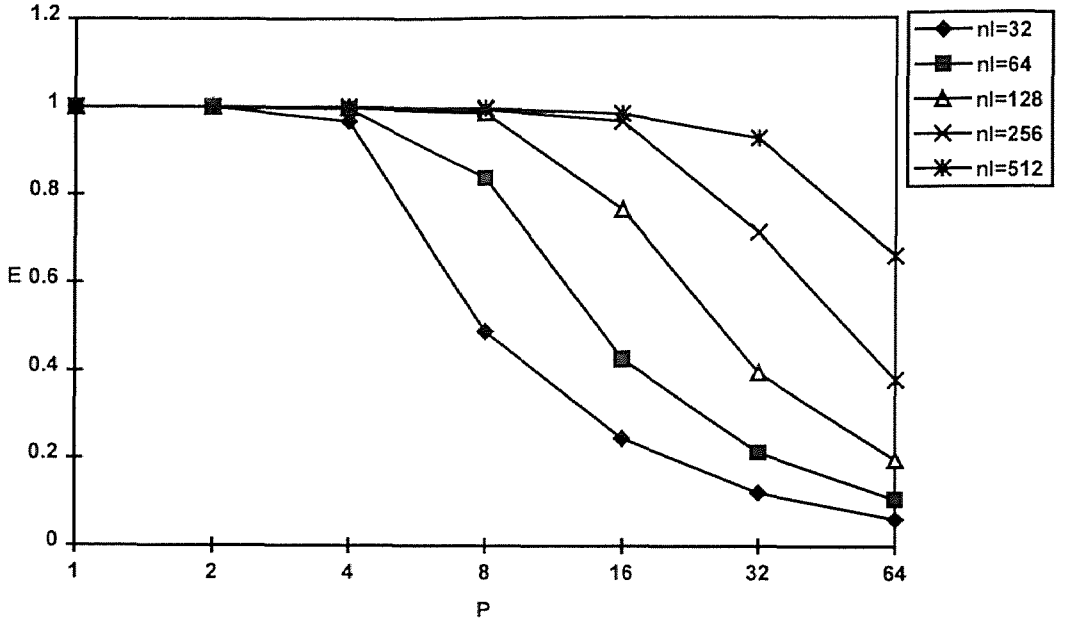

Figure 4.7 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=3x3

Table 4.8 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=3x3

Speedup vs. Shared-bus-width, *template matrix*=3x3							
nl	SP1	SP2	SP4	SP8	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.1470	1.1472	1.1806	1.9628	1.9982	1.9982	2.0028
128	1.2380	1.2390	1.2775	2.4942	3.8654	3.9963	3.9994
256	1.2891	1.2896	1.3319	2.6168	5.0822	7.5116	7.9988
512	1.3163	1.3168	1.3584	2.6828	5.2807	9.9693	14.2123

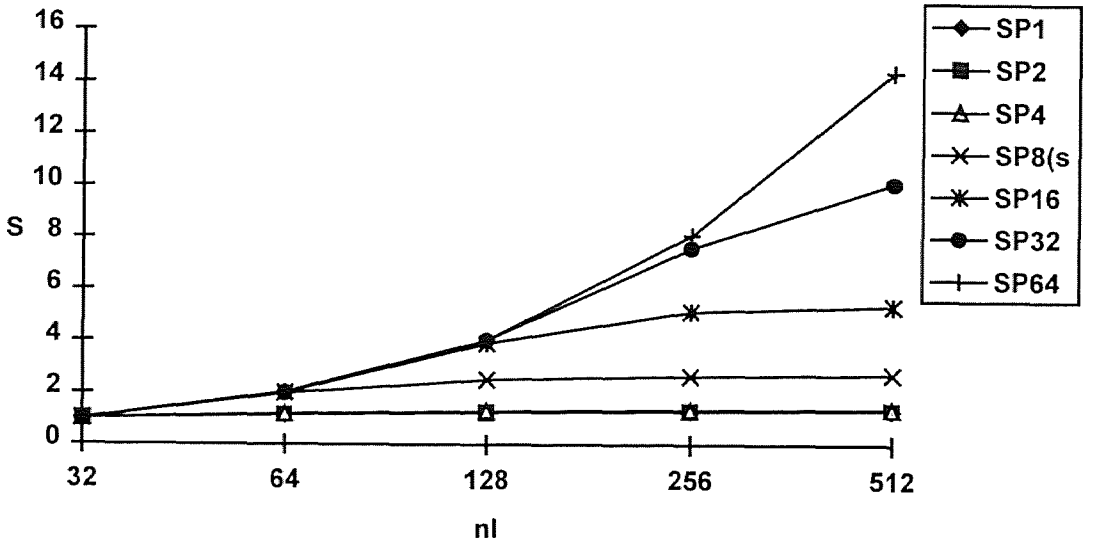

Figure 4.8 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=3x3

4.3 Effect of Shared-Bus Width on Performance when Template Matrix =6x6

Tables 4.9, 4.10 and 4.11 and Figures 4.9, 4.10 and 4.11 show that Convolution with a *template matrix* of 6x6 is effectively executed (E≥0.50) by up to *16* processors if *nl*=32, by up to *32* processors if *nl*=64, by up to *64* processors if *nl*=128, 256, and 512.

Figures 4.10 and 4.11 and Tables 4.10 and 4.11 show near perfect speedup factors and system efficiencies (E≥0.90) for *nl*=32&P=2,4,8, for *nl*=64&P=2,4,8,16, for *nl*=128&P=2,4,8,16,32, for *nl*=256&P=2,4,8,16, 32, for *nl*=512&P=2,4,8,16,32,64.

Table 4.12 and Figure 4.12 show remarkable speedups of specific P-node systems, which are due only to *nl* increase. In Appendix B, Table B.5 and Figures B.7 and B8 show, in more detail, how the execution time decreases by increasing the shared-bus-width. Observing, for example, the first (0.680 *sec*) and the last (0.129 *sec*) entry of the last column (corresponding to the 64-node system) in Table B.5, we measure a *shared-bus speedup factor* of 5.27. The value 5.27 can be confirmed by looking up the entry with coordinates *nl*=512&P=64 in Table 4.12. Table 4.12 and Figure 4.12 show a maximum *shared-bus-speedup-factor* of 1.1, 1.1, 1.1, 1.1, 1.5, 2.8 and 5.3 for 1-node, 2-node, 4-node, 8-node, 16-node, 32-node, and 64-node systems respectively.

Table 4.9 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=6x6

Time vs. Number of Processors, *template matrix*=6x6					
	nl=32	*nl*=64	*nl*=128	*nl*=256	*nl*=512
P	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
1	8.2323	7.8916	7.7212	7.6360	7.5934
2	4.1167	3.9460	3.8607	3.8181	3.7967
4	2.0642	1.9760	1.9318	1.9098	1.8987
8	1.0460	0.9949	0.9694	0.9566	0.9502
16	0.7021	0.5124	0.4922	0.4820	0.4770
32	0.6801	0.3610	0.2615	0.2487	0.2423
64	0.6801	0.3401	0.1903	0.1401	0.1290

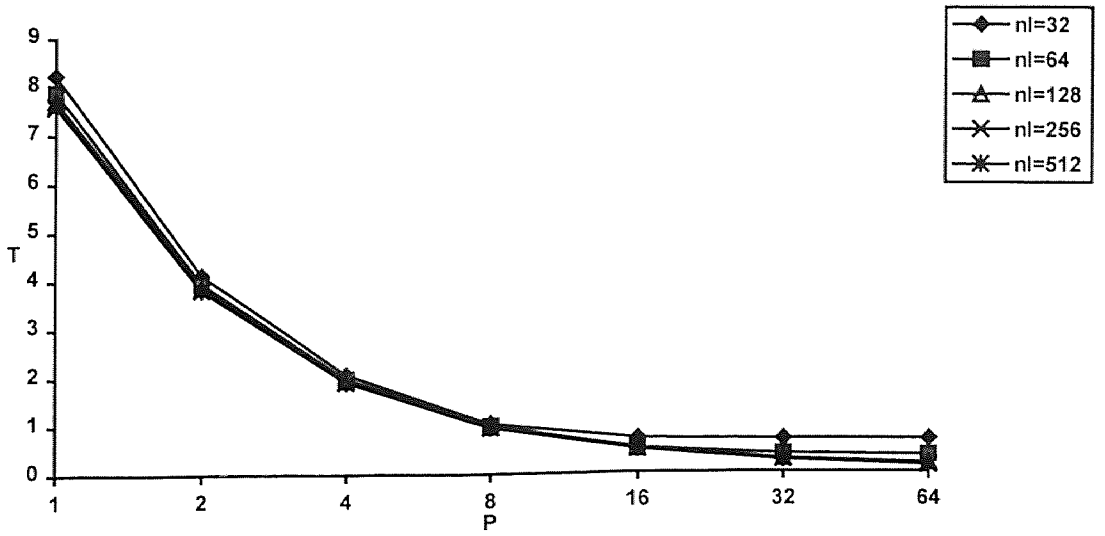

Figure 4.9 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=6x6

Table 4.10 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=6x6

Speedup vs. Number of Processors, *template matrix*=6x6					
	S	**S**	**S**	**S**	**S**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.9997	1.9999	1.9999	1.9999	2.0000
4	3.9881	3.9937	3.9969	3.9983	3.9993
8	7.8703	7.9321	7.9649	7.9824	7.9914
16	11.7253	15.4012	15.6871	15.8423	15.9191
32	12.1045	21.8604	29.5266	30.7037	31.3388
64	12.1045	23.2038	40.5738	54.5039	58.8636

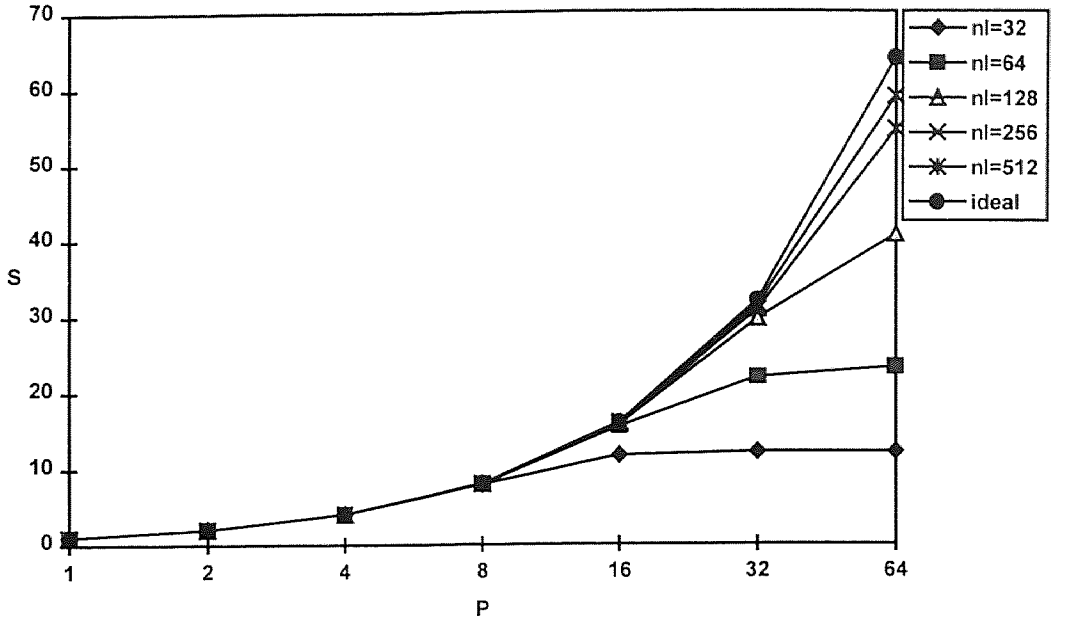

Figure 4.10 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=6x6

Table 4.11 Efficiency vs. P for *nl*=32,64,128,256 and 512 when *template matrix*=6X6

Efficiency vs. Number of Processors, *template matrix*=6X6					
	E	**E**	**E**	**E**	**E**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9999	0.9999	1.0000	1.0000	1.0000
4	0.9970	0.9984	0.9992	0.9996	0.9998
8	0.9838	0.9915	0.9956	0.9978	0.9989
16	0.7328	0.9626	0.9804	0.9901	0.9949
32	0.3783	0.6831	0.9227	0.9595	0.9793
64	0.1891	0.3626	0.6340	0.8516	0.9197

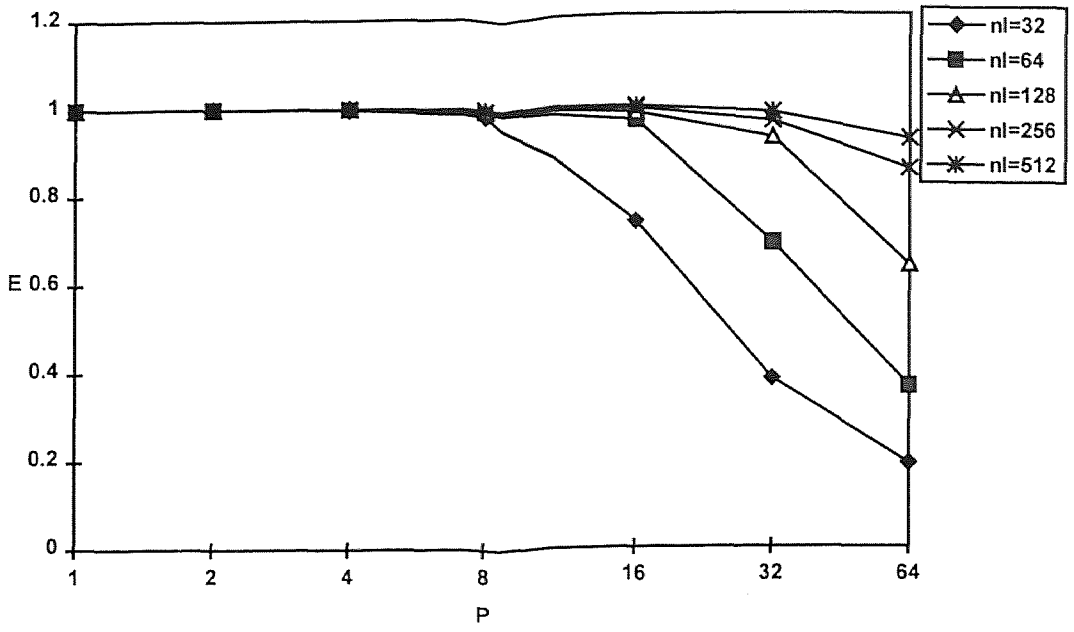

Figure 4.11 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=6X6

Table 4.12 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

Speedup vs. Shared-bus-width, *template matrix*=6x6							
nl	SP1	SP2	SP4	SP8(s	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.0431	1.0433	1.0445	1.0481	1.3711	1.8889	2.0000
128	1.0662	1.0666	1.0683	1.0795	1.4268	2.6054	3.5789
256	1.0781	1.0783	1.0806	1.0930	1.4564	2.7419	4.8571
512	1.0842	1.0843	1.0869	1.1011	1.4748	2.8099	5.2713

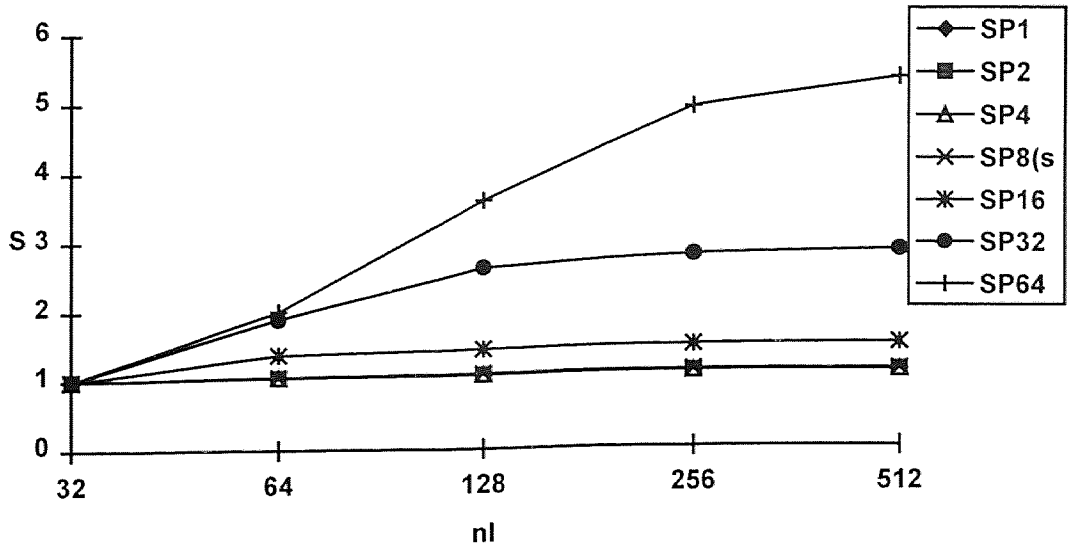

Figure 4.12 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

4.4 Effect of Shared-Bus Width on Performance when Template Matrix =9x9

Tables 4.13, 4.14 and 4.15 and Figures 4.13, 4.14 and 4.15 show that Convolution with a *template matrix* of 9x9 is effectively executed (E≥0.50) by up to 32 processors if *nl*=32 by up to 64 processors if *nl*=64, 128, 256 and 512.

Figures 4.14 and 4.15 and Tables 4.14 and 4.15 show near perfect speedup factors and system efficiencies (E≥0.90) for *nl*=32&P=2,4,8,16, for *nl*=64&P=2,4,8,16,32, for *nl*=128&P=2,4,8,16,32, for *nl*=256&P=2,4,8,16,32,64, for *nl*=512&P=2,4,8,16,32,64.

Table 4.16 and Figure 4.16 show speedups of specific P-node systems, which are due only to *nl* increase. In Appendix B, Table B.7 and Figures B.10 and B.11 show, in more detail, how the execution time decreases by increasing the shared-bus-width.

Observing, for example, the first (0.710 *sec*) and the last (0.277 *sec*) entry of the last column (corresponding to the 64-node system) in Table B.7, we measure a *shared-bus-speedup-factor* of 2.56. The value 2.56 can be confirmed by looking up the entry with coordinates *nl*=512&P=64 in Table 4.16. Table 4.16 and Figure 4.16 show a maximum a *shared-bus-speedup-factor* of 1.04, 1.04, 1.04, 1.04, 1.07, 1.43 and 2.56 for 1-node, 2-node, 4-node, 8-node, 16-node, 32-node, and 64-node systems respectively.

The *shared-bus-speedup-factor* when *template window*=9x9 is not significant. The reason is that applications incorporating very large amounts of spatial and temporal localities (like the one we examine) demand much less shared-bus bandwidth compared to applications with less spatial and temporal localities. If there is enough supply of shared-bus bandwidth when *nl*=32, we do not expect a significant improvement of performance when a wider shared-bus is utilized. How spatial and temporal locality affects the performance of the *twin-prefetching* multiprocessor is examined thoroughly in Chapter 5.

Table 4.13 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Time vs Number of Processors, *template matrix*=9x9					
	nl=32	*nl*=64	*nl*=128	*nl*=256	*nl*=512
P	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
1	17.7007	17.3445	17.1664	17.0774	17.0328
2	8.8508	8.6724	8.5833	8.5387	8.5164
4	4.4316	4.3393	4.2932	4.2701	4.2586
8	2.2303	2.1769	2.1502	2.1369	2.1302
16	1.1465	1.1040	1.0829	1.0723	1.0670
32	0.7674	0.5843	0.5576	0.5442	0.5376
64	0.7109	0.4108	0.3116	0.2885	0.2770

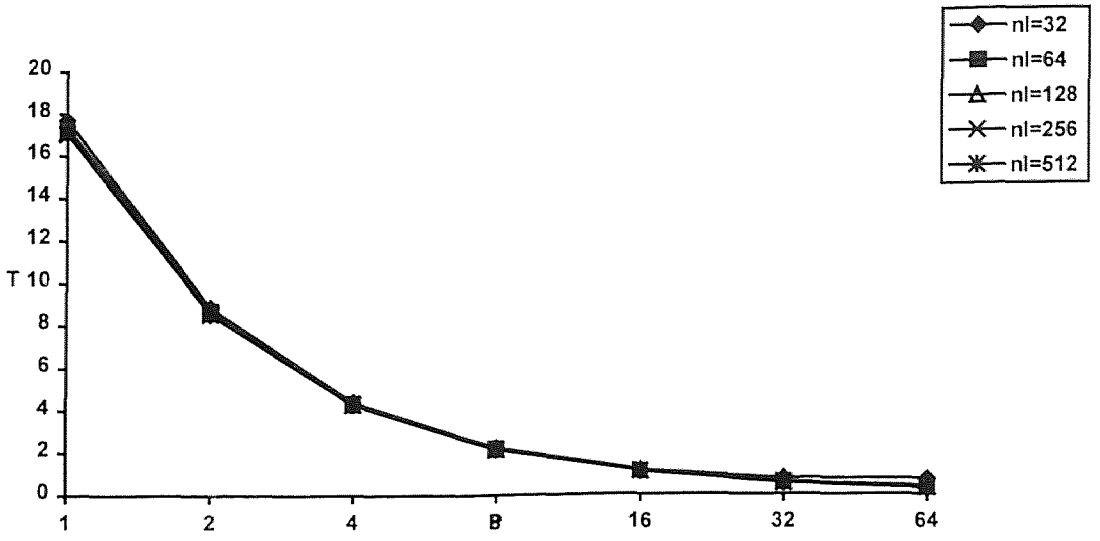

Figure 4.13 Time vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Table 4.14 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Speedup vs Number of Processors, *template matrix*=9x9					
	S	**S**	**S**	**S**	**S**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.9999	2.0000	2.0000	2.0000	2.0000
4	3.9942	3.9971	3.9985	3.9993	3.9996
8	7.9365	7.9675	7.9836	7.9917	7.9959
16	15.4389	15.7106	15.8522	15.9260	15.9633
32	23.0658	29.6842	30.7862	31.3807	31.6830
64	24.8990	42.2213	55.0911	59.1938	61.4903

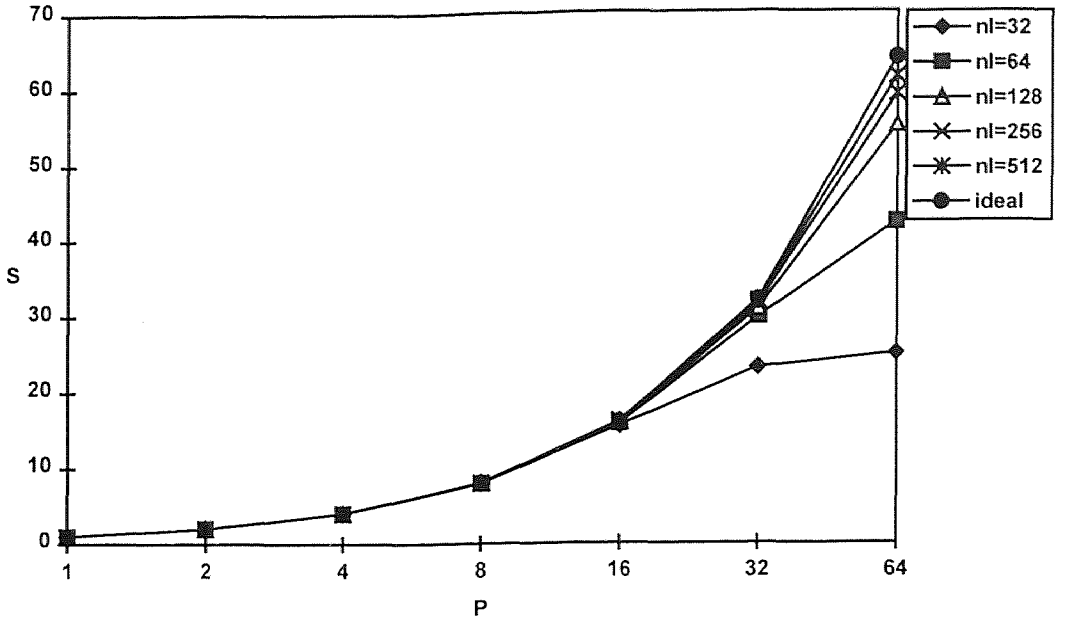

Figure 4.14 Speedup vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Table 4.15 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Efficiency vs. Number of Processors, *template matrix*=9X9					
	E	**E**	**E**	**E**	**E**
P	***nl*=32**	***nl*=64**	***nl*=128**	***nl*=256**	***nl*=512**
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9999	1.0000	1.0000	1.0000	1.0000
4	0.9986	0.9993	0.9996	0.9998	0.9999
8	0.9921	0.9959	0.9980	0.9990	0.9995
16	0.9649	0.9819	0.9908	0.9954	0.9977
32	0.7208	0.9276	0.9621	0.9806	0.9901
64	0.3890	0.6597	0.8608	0.9249	0.9608

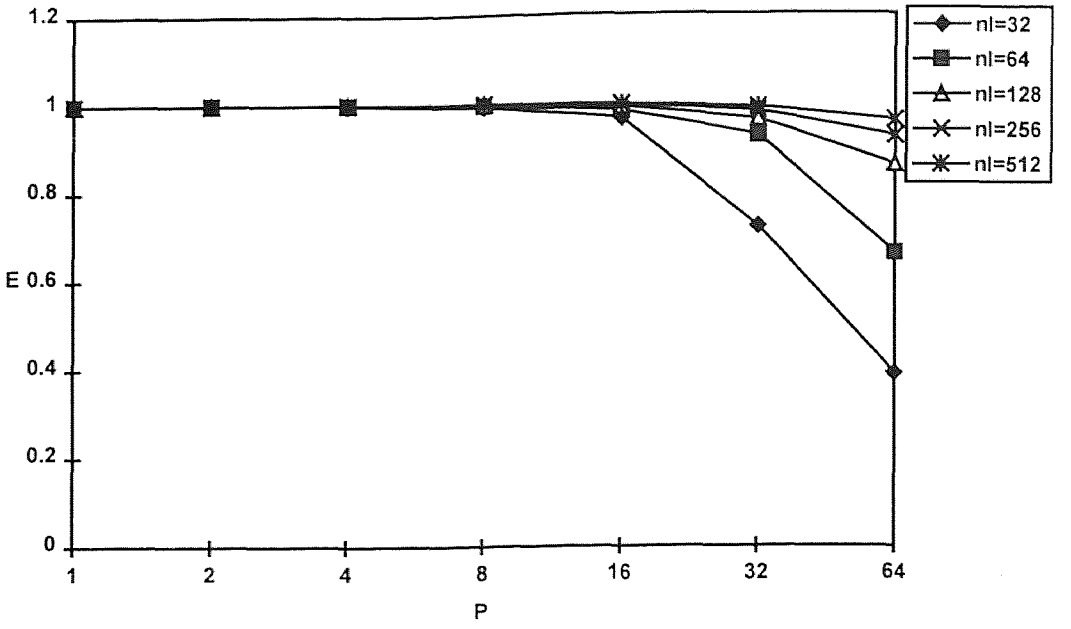

Figure 4.15 Efficiency vs. P for *nl*=32, 64, 128, 256 and 512 when *template matrix*=9x9

Table 4.16 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

Speedup vs. Shared-bus-width, *template matrix*=9x9							
nl	SP1	SP2	SP4	SP8	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.0206	1.0206	1.0214	1.0243	1.0380	1.3134	1.7317
128	1.0312	1.0312	1.0324	1.0372	1.0582	1.3770	2.2830
256	1.0365	1.0367	1.0379	1.0435	1.0690	1.4099	2.4653
512	1.0392	1.0393	1.0409	1.0469	1.0740	1.4283	2.5632

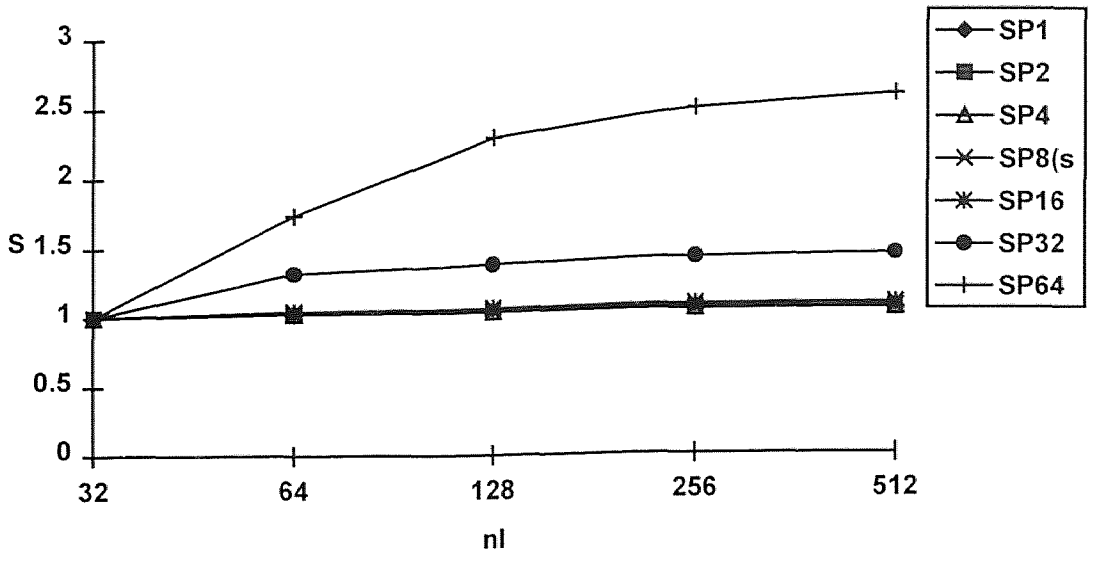

Figure 4.16 Speedup vs. Shared-bus-width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

4.5 Discussion of Results

The elimination of the traditional direct linkage of the shared-bus and processor data bus makes feasible the utilization of a wider shared-bus. Additional bandwidth provided by a wider shared-bus along with *twin-prefetching* mechanism makes possible the effective support (E>0.50) of 32 processors, and the near perfect effective support (E>0.90) of 16 processors. The above numbers of effectively supported processors are based on the worst case scenario when an application embedding a small amount of spatial and temporal locality *(template matrix*-2x2) is executed. In the best case scenario, when an application embedding very high spatial and temporal locality *(template matrix*=9x9) is executed, even a 64-node *twin-prefetching* multiprocessor achieves a *system efficiency* greater than 0.9 (E>0.90).

Tables 4.17 shows how many processors can be effectively supported (E>0.50) for every value of *nl*. For example, the entry in Table 4.17 with coordinates nl=512&template matrix=3x3, reads 2,4,8,16,32,64. This means that the *twin-prefetching* system utilizing a shared-bus with 512 data lines could effectively support (E>0.50) up to 64 processors.

Tables 4.18 shows the P-node *twin-prefetching* multiprocessors achieving nearly perfect system efficiency (E>0.90). Table 4.18 indicates a very large number of P-node systems achieving such a high system efficiency value.

Tables 4.17 and 4.18 indicate which systems should or should not be implemented. Let us take as an example the first entry (2, 4, 8, 16, 32) of the column labeled nl=512 in Table 4.17. Since the number 64 is not included in the entry a 64-node *twin-prefetching* system should not be built (because E<0.5) if an application with similar characteristics (*Twratio*) to the one indicated (2x2) is run on the system. Depending on the system efficiency required we select the appropriate table as a guide for the design. It is important to note that if a specific P-node *twin-prefetching* system if effectively supported by more than one value of *nl*, we should choose the smallest value. As an example let us take the values of the first row of Table 4.18. We observe that shared-bus-width of 64 or 128 or 256 or 512 supports well (E>0.90) a 4-node system. The logical choice among all shared-bus-widths is 64.

Finally, the effect of utilizing a wider shared-bus for a *twin-prefetching* multiprocessor is clearly demonstrated in both Tables 4.17 and 4.18, i.e., a wider shared-bus increases significantly the number of effectively supported processing elements.

Table 4.17 P-node twin-prefetching multiprocessors with system efficiency E>0.50

tmpl.mtrx	nl=32	nl=64	nl=128	nl=256	nl=512
2x2	2,4	2,4	2,4,8	2,4,8,16	2,4,8,16,32
3x3	2,4	2,4,8	2,4,8,16	2,4,8,16,32	2,4,8,16,32, 64
6x6	2,4,8,16	2,4,8,16,32	2,4,8,16,32, 64	2,4,8,16,32, 64	2,4,8,16,32, 64
9x9	2,4,8,16,32	2,4,8,16,32, 64	2,4,8,16,32, 64	2,4,8,16,32, 64	2,4,8,16,32, 64

Table 4.18 P-node twin-prefetching multiprocessors with system efficiency E>0.90

tmpl. mtrx	nl=32	nl=64	nl=128	nl=256	nl=512
2x2	2	2,4	2,4	2,4,8	2,4,8,16
3x3	2,4	2,4	2,4,8	2,4,8,16	2,4,8,16,32
6x6	2,4,8	2,4,8,16	2,4,8,16,32	2,4,8,16,32	2,4,8,16,32, 64
9x9	2,4,8,16	2,4,8,16,32	2,4,8,16,32	2,4,8,16,32, 64	2,4,8,16,32, 64

CHAPTER 5

EFFECT OF DATA LOCALITY ON PERFORMANCE

This chapter investigates the effect of spatial and temporal locality of data on the performance of the proposed twin-prefetching shared-memory shared-bus multiprocessor system. The performance of any computer system is greatly affected by the amount of data locality present in an application. Traditional caches were invented to take advantage of data locality and increase performance. (Spatial locality is the tendency of a program to access items whose addresses are near one another i.e., indicates that if a location in memory is accessed, then others nearby will probably be accessed soon. Temporal locality denotes that if an address in memory is accessed once, then it will probably be accessed again soon.) The proposed system employs software-controlled prefetching in caches; a technique that further increases performance according to recent research.

In the case of the convolution algorithm, the number of local neighborhood operations is a direct measure of the amount of data locality. The following discussion intends to show the correlation of data locality, size of template matrix and number of memory references of every pixel value in memory. The number of local neighborhood operations is equivalent to the number of coefficients in the template matrix by definition. In the case of the proposed system, the number of local neighborhood operations is also equivalent to the number of memory references required to produce an output pixel, i.e., due to the dual data fetch capability of the ADSP-21060, (other modern DSPs have similar capabilities), a coefficient and an input pixel value are fetched simultaneously

from memory; therefore, for a template matrix consisting of k coefficients, k dual fetches (one from internal memory and one from external memory) take place to produce an output pixel. We conclude that in the case of the convolution algorithm every datum (except the edges) of an image segment transferred in the prefetching cache is referenced k times (temporally or spatially). From this point on, locality k of an application will denote that the average number of times which every image pixel is referenced is k.

Our goal of investigating the proposed system on different amounts of data locality is achieved by increasing or decreasing the size of the template matrix. Specifically, template matrix size is varied from 2x2 to 3x3 to 6x6 to 9x9; therefore, the number of memory references of every image pixel value in data memory is 4, 9, 36 and 81, respectively. Thus, a wide range of data locality cases are covered.

There are *five* tables labeled "Execution Time vs. Number of Processors," one table for every value that the *shared-bus-width* receives (*nl*=32, 64, 128, 256, 512). Each table consists of *five* columns. The first column contains P values, while the other *four* columns contain timings of the P-node system when the *template matrix* used has size 2x2, 3x3, 6x6 and 9x9. The corresponding speedup S(P), and efficiency E(P), of every entry in the tables labeled "Execution Time vs. Number of Processors" are found in tables labeled "Speedup vs. Number of Processors" and "Efficiency vs. Number of Processors." Notation in this chapter for tables or figures is as follows: *Tmpl.mtrx* or *mtrx* stands for *template matrix*, S(*zxz*) and E(*zxz*) stand for *speedup* and *efficiency,* respectively, when a *template matrix* of size *zxz* is used, P stands for number of processors, *nl* stands for shared-bus-width and 1w/o stands for one processor without the twin-prefetching caches.

An effective system is considered to be a system with a speedup factor greater than P/2, i.e., providing system efficiency greater than 0.5. Shaded gray areas within the tables point out effective systems.

5.1 Effect of Spatial and Temporal Locality on a P-Node Twin-Prefetching Multiprocessor when Shared-Bus-Width is 32

Tables 5.1, 5.2, 5.3, and Figures 5.2, 5.3, 5.4 show how the proposed twin-prefetching system performs as the size of *template matrix* (amount of data locality) varies, when shared-bus-width is 32. Adjacent row entries in Table 5.1 report two timings that give us an indication of how fast two system configurations, employing different number of processors, execute the same application. A useful piece of information is obtained by observing the location at each column where the values of adjacent rows do not differ significantly. The P coordinate at this location denotes the maximum number of processors, which can effectively (with system efficiency greater that 50%) execute the application. It should be mentioned that this location is the last gray shaded row in a column. From left to right column shaded areas widen, denoting a sharp increase of the number of processors that can effectively execute the application. When a 2x2 template matrix (locality 4) is used only 4 processors effectively execute the application, while if a 9x9 template matrix (locality 81) is used the number of processors increases to 32. The fuel that drives the additional 28 processors is data locality. Because of greater data locality a larger number of references, for every image pixel in the prefetching cache, keeps processors occupied with the same image segment for a longer period of time, thus allowing a greater number of processors to be serviced by the shared-bus.

Tables 5.2 and 5.3, and Figures 5.3 and 5.4 show the speedup and efficiency of the system for different sizes of template matrix and demonstrate the significant change of performance depending on data locality. Speedup and efficiency are better system performance evaluators (because they are observer friendly); a good speedup value approaches value P while a good system efficiency approaches 1. When, for example, P=16 in Table 5.2, the speedup is 2.31 for template matrix=2x2 while the speedup is 15.43 for template matrix=9x9. Likewise, when P=16 in Table 5.3, system efficiency is 0.14 for template matrix=2x2 while efficiency is 0.96 for template matrix=9x9. The significant improvement in system efficiency is solely due to a greater amount of data locality since all other software and hardware parameters are kept constant.

5.1.1 Performance of One DSP Processor Without Twin-Prefetching Cache Memories

Twin-prefetching caches were removed from the one node system (P=1) and all four applications were executed assuming all image data (eight images of size 1024x1024) stored in main memory. Figure 5.1 shows a graph comparing the execution times of one node system with and one without the prefetching caches. All applications were executed faster on the system employing twin-prefetching. Results show a speedup of 2.88, 2.75, 2.23, and 1.70 when the template matrix used is 9x9, 6x6, 3x3, and 2x2, respectively. Therefore applications employing greater amount of data locality achieve better speedup. Also it is demonstrated that twin-prefetching could benefit uniprocessor systems too.

P-node systems (P>1), without the twin-prefetching caches, were not simulated because a shared-bus would not be able to support more than one ADSP-21060. It should be noted that ADSP-21060 does not carry on-chip cache.

Table 5.1 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

Execution Time *vs.* Number of Processors when *nl*=32				
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1w/o	2.5176	5.6633	22.6502	50.9618
1	1.4803	2.5391	8.2323	17.7007
2	0.7413	1.2701	4.1167	8.8508
4	0.6393	0.6569	2.0642	4.4316
8	0.6393	0.6495	1.0460	2.2303
16	0.6393	0.6495	0.7021	1.1465
32	0.6393	0.6495	0.6801	0.7674
64	0.6393	0.6495	0.6801	0.7109

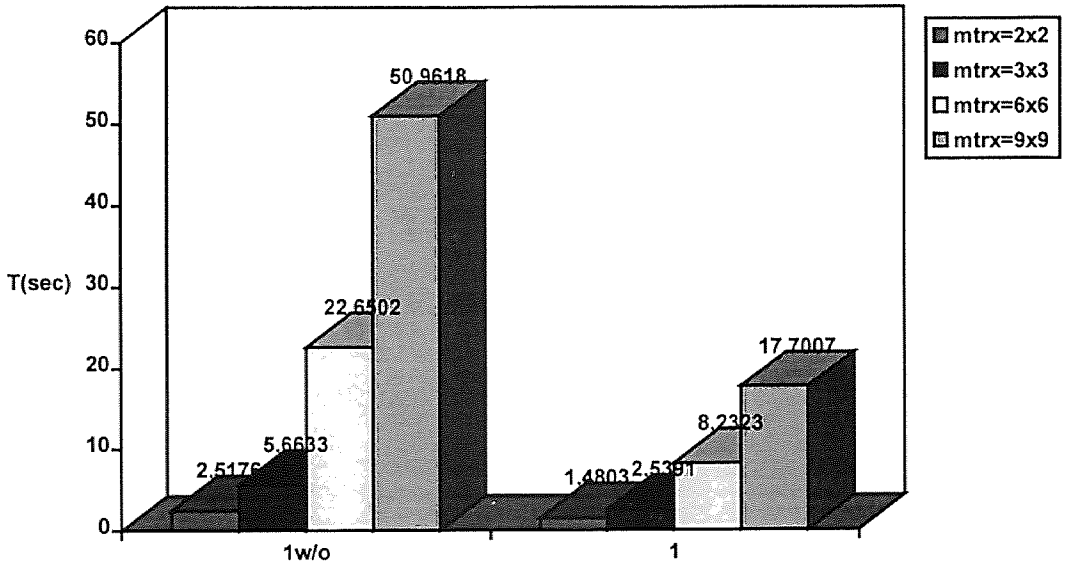

Figure 5.1 Execution time of applications on a uniprocessor system with and without twin prefetching (*nl*=32)

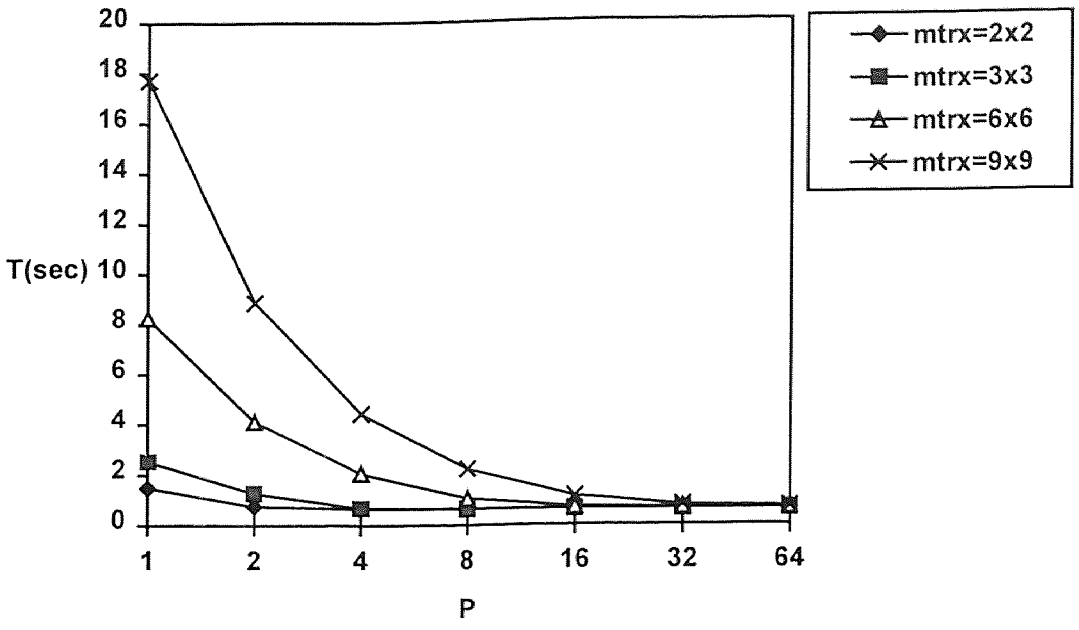

Figure 5.2 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

Table 5.2 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

Speedup vs. Number of Processors, *nl*=32				
P	S(2x2)	S(3x3)	S(6x6)	S(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	1.9969	1.9991	1.9997	1.9999
4	2.3155	3.8653	3.9881	3.9942
8	2.3155	3.9093	7.8703	7.9365
16	2.3155	3.9093	11.7253	15.4389
32	2.3155	3.9093	12.1045	23.0658
64	2.3155	3.9093	12.1045	24.8990

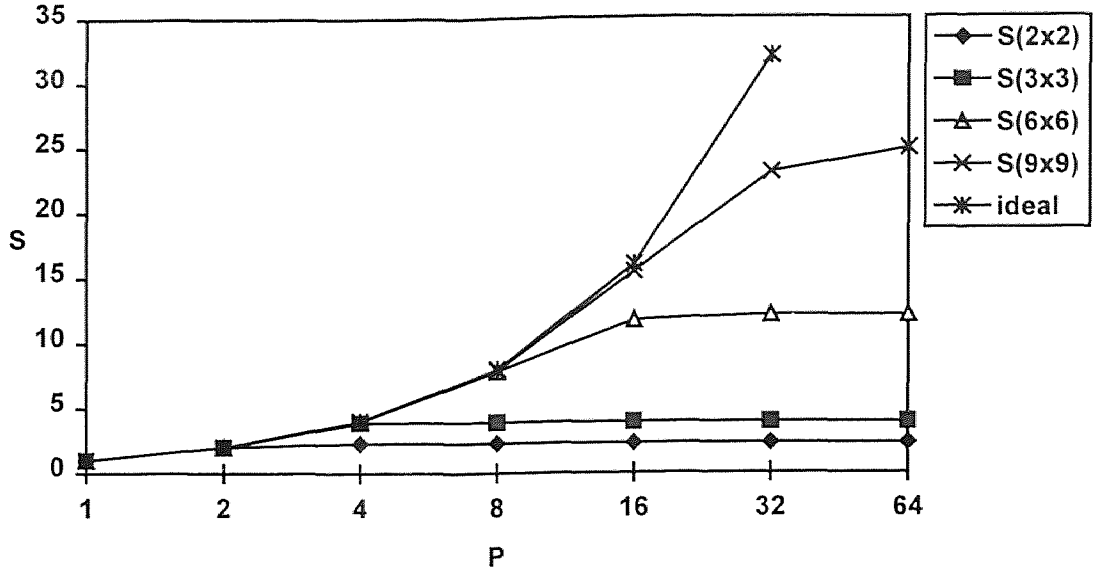

Figure 5.3 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

Table 5.3 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

Efficiency vs. Number of Processors, *nl*=32				
P	E(2x2)	E(3x3)	E(6x6)	E(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	0.9984	0.9996	0.9999	0.9999
4	0.5789	0.9663	0.9970	0.9986
8	0.2894	0.4887	0.9838	0.9921
16	0.1447	0.2443	0.7328	0.9649
32	0.0724	0.1222	0.3783	0.7208
64	0.0362	0.0611	0.1891	0.3890

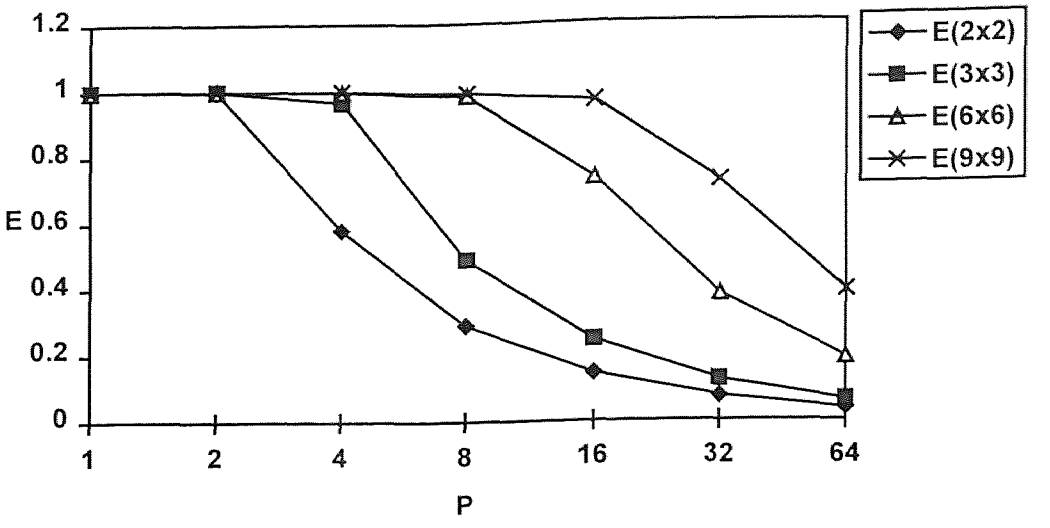

Figure 5.4 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=32

5.2 Effect of Spatial and Temporal Locality on a P-Node Twin-Prefetching Multiprocessor when Shared-Bus-Width is 64

Tables 5.4, 5.5, 5.6, and Figures 5.5, 5.6, 5.7 show how the proposed twin-prefetching system performs as the size of *template matrix* (amount of data locality) varies, when shared-bus-width is 64. From left to right column shaded areas widen, denoting a sharp increase of the number of processors that can effectively execute the application. When a 2x2 template matrix (locality 4) is used only 4 processors effectively execute the application while if a 9x9 template matrix (locality 81) is used the number of processors increases to 64. The fuel that drives the additional 60 processors is data locality. Because of greater data locality a larger number of references, for every image pixel in the prefetching cache, keeps processors occupied with the same image segment for a longer

period of time; thus allowing a greater number of processors to be serviced by the shared-bus.

Tables 5.5 and 5.6, and Figures 5.6 and 5.7 show the speedup and efficiency of the system for different sizes of template matrix and demonstrate the significant change of performance depending on data locality. When, for example, P=32 in Table 5.5 speedup is 3.62 for template matrix=2x2 while speedup is 29.68 for template matrix=9x9. Likewise, when P=32 in Table 5.6 system efficiency is 0.11 for template matrix=2x2 while efficiency is 0.92 for template matrix=9x9. The significant improvement in system efficiency is solely due to the greater amount of data locality since all other software and hardware parameters are kept constant.

Table 5.4 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

Execution Time vs. Number of Processors when *nl*=64				
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	1.1600	2.2137	7.8916	17.3445
2	0.5803	1.1071	3.9460	8.6724
4	0.3226	0.5564	1.9760	4.3393
8	0.3197	0.3309	0.9949	2.1769
16	0.3197	0.3248	0.5124	1.1040
32	0.3197	0.3248	0.3610	0.5843
64	0.3197	0.3248	0.3401	0.4108

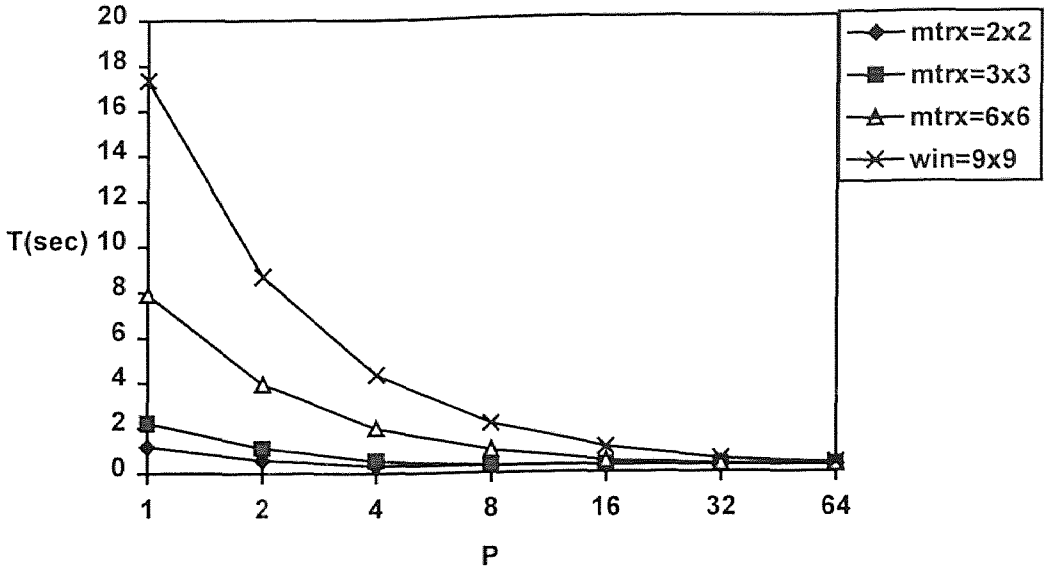

Figure 5.5 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

Table 5.5 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

Speedup vs. Number of Processors when, *nl*=64				
P	S(2x2)	S(3x3)	S(6x6)	S(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	1.9990	1.9995	1.9999	2.0000
4	3.5958	3.9786	3.9937	3.9971
8	3.6284	6.6899	7.9321	7.9675
16	3.6284	6.8156	15.4012	15.7106
32	3.6284	6.8156	21.8604	29.6842
64	3.6284	6.8156	23.2038	42.2213

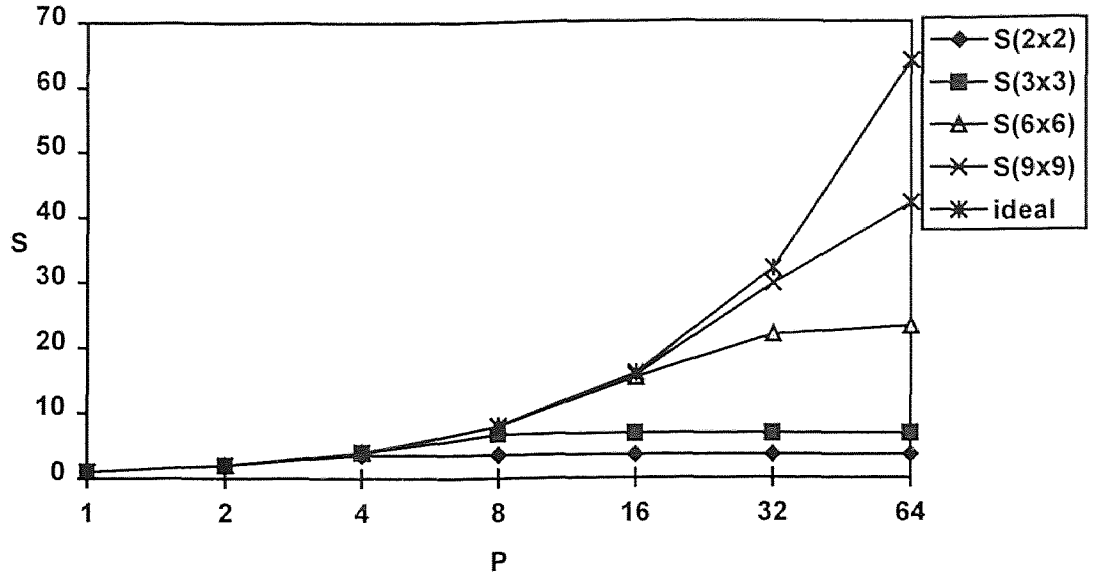

Figure 5.6 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

Table 5.6 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

Efficiency vs. Number of Processors, *nl*=64				
P	E(2x2)	E(3x3)	E(6x6)	E(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	0.9995	0.9998	0.9999	1.0000
4	0.8989	0.9947	0.9984	0.9993
8	0.4536	0.8362	0.9915	0.9959
16	0.2268	0.4260	0.9626	0.9819
32	0.1134	0.2130	0.6831	0.9276
64	0.0567	0.1065	0.3626	0.6597

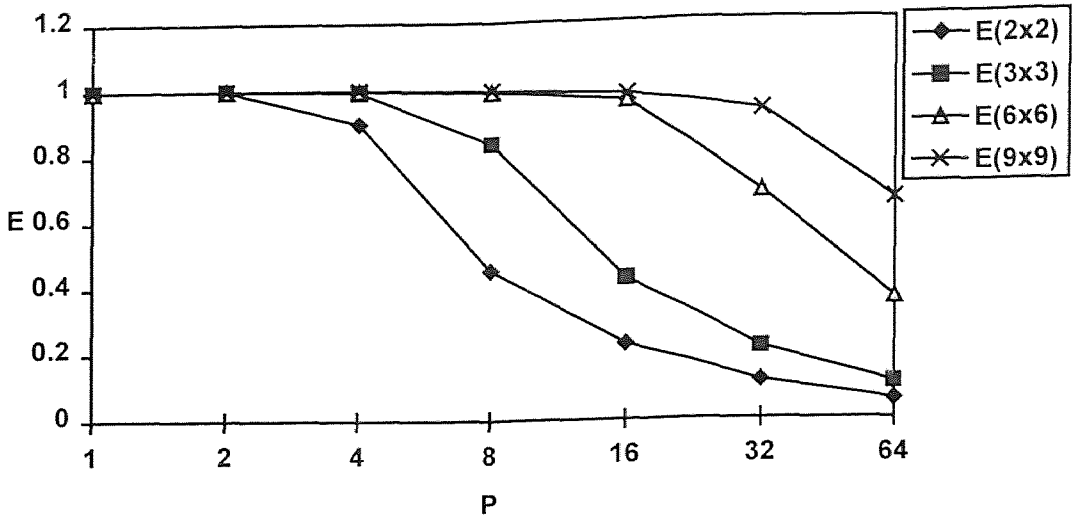

Figure 5.7 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=64

5.3 Effect of Spatial and Temporal Locality on a P-Node Twin-Prefetching Multiprocessor when Shared-Bus-Width is 128

Tables 5.7, 5.8 and 5.9, and figures 5.8, 5.9 and 5.10 show how the proposed twin-prefetching system performs as the size of *template matrix* (amount of data locality) varies, when shared-bus-width is 128. From left to right column shaded areas widen, denoting a sharp increase of the number of processors that can effectively execute the application. When a 2x2 template matrix (locality 4) is used 8 processors effectively execute the application while if a 9x9 template matrix (locality 81) is used the number of processors increases to 64. The fuel that drives the additional 56 processors is data locality. Because of greater data locality a larger number of references, for every image

pixel in the prefetching cache, keeps processors occupied with the same image segment for a longer period of time, thus allowing a greater number of processors to be serviced by the shared-bus.

Tables 5.8 and 5.9, and figures 5.9 and 5.10 show the speedup and efficiency of the system for different sizes of template matrix and demonstrate the significant change of performance depending on data locality. When, for example, P=32 in Table 5.8, speedup is 6.25 for template matrix=2x2 while speedup is 30.78 for template matrix=9x9. Likewise, when P=32 in Table 5.9 system efficiency is 0.19 for template matrix=2x2 while efficiency is 0.96 for template matrix=9x9. The significant improvement in system efficiency is solely due to the greater amount of data locality since all other software and hardware parameters are kept constant.

Table 5.7 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

Execution Time vs. Number of Processors when *nl*=128				
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	0.9999	2.0510	7.7212	17.1664
2	0.5001	1.0257	3.8607	8.5833
4	0.2514	0.5142	1.9318	4.2932
8	0.1621	0.2604	0.9694	2.1502
16	0.1599	0.1679	0.4922	1.0829
32	0.1599	0.1624	0.2615	0.5576
64	0.1599	0.1624	0.1903	0.3116

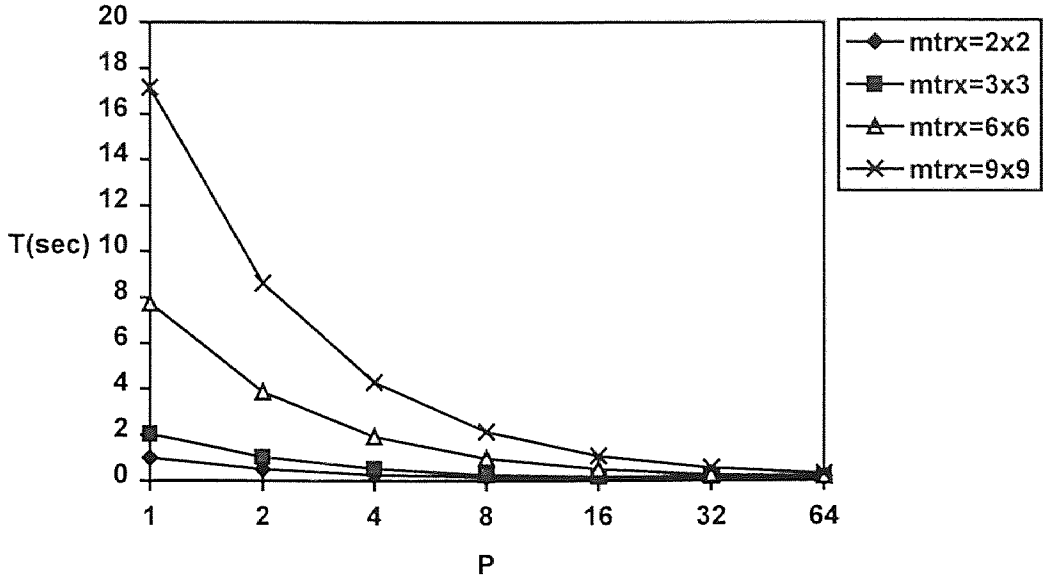

Figure 5.8 Execution Time *vs.* Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

Table 5.8 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

Speedup vs. Number of Processors, *nl*=128				
P	S(2x2)	S(3x3)	S(6x6)	S(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	1.9994	1.9996	1.9999	2.0000
4	3.9773	3.9887	3.9969	3.9985
8	6.1684	7.8763	7.9649	7.9836
16	6.2533	12.2156	15.6871	15.8522
32	6.2533	12.6293	29.5266	30.7862
64	6.2533	12.6293	40.5738	55.0911

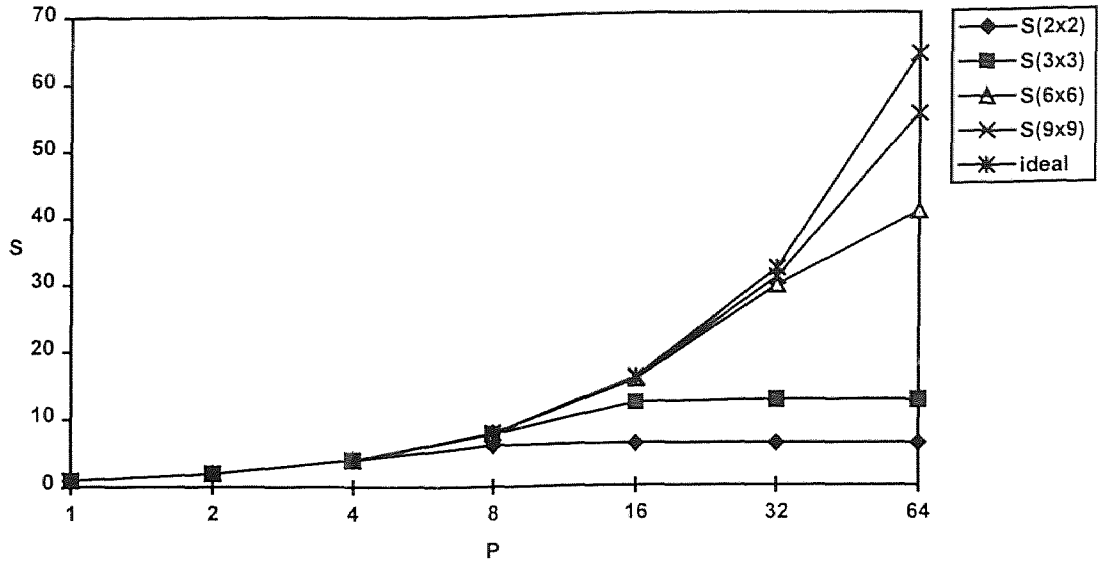

Figure 5.9 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

Table 5.9 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

Efficiency vs. Number of Processors, *nl*=128				
P	E(2x2)	E(3x3)	E(6x6)	E(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	0.9997	0.9998	1.0000	1.0000
4	0.9943	0.9972	0.9992	0.9996
8	0.7711	0.9845	0.9956	0.9980
16	0.3908	0.7635	0.9804	0.9908
32	0.1954	0.3947	0.9227	0.9621
64	0.0977	0.1973	0.6340	0.8608

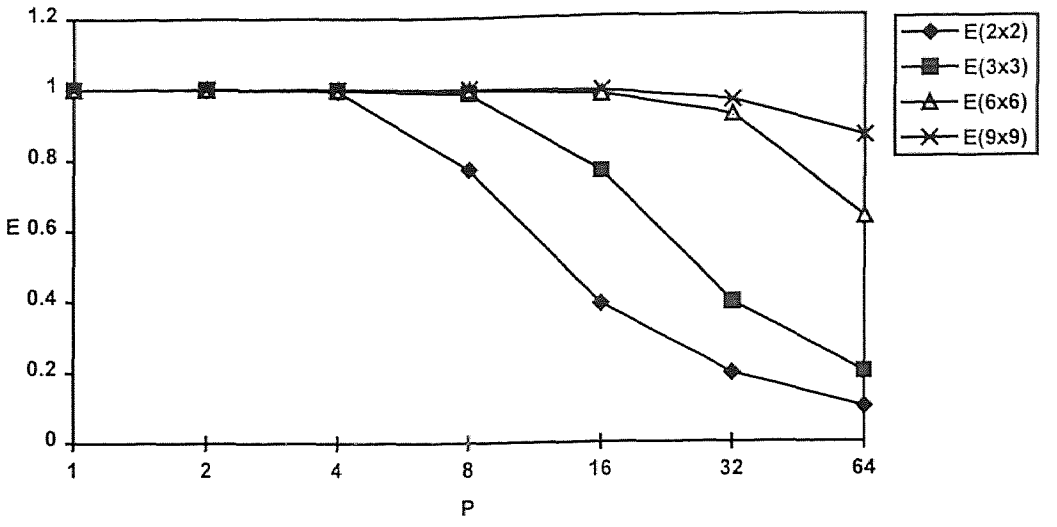

Figure 5.10 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=128

5.4 Effect of Spatial and Temporal Locality on a P-Node Twin-Prefetching Multiprocessor when Shared-Bus-Width is 256

Tables 5.10, 5.11 and 5.12, and figures 5.11, 5.12 and 5.13 show how the proposed twin-prefetching system performs as the size of *template matrix* (amount of data locality) varies, when shared-bus-width is 256. From left to right column shaded areas widen, denoting a sharp increase of the number of processors that can effectively execute the application. When a 2x2 template matrix (locality 4) is used, 16 processors effectively execute the application while if a 9x9 template matrix (locality 81) is used the number of processors increases to 64. The fuel that drives the additional 48 processors is data

locality. Because of greater data locality, a larger number of references, for every image pixel in the prefetching cache, keeps processors occupied with the same image segment for a longer period of time; thus allowing a greater number of processors to be serviced by the shared-bus.

Tables 5.11 and 5.12, and Figures 5.12 and 5.13 show the speedup and efficiency of the system for different sizes of template matrix and demonstrate the significant change of performance depending on data locality. When, for example, P=32 in Table 5.11, the speedup reached is 11.49 for template matrix=2x2 while speedup is 31.38 for template matrix=9x9. Likewise, when P=32 in Table 5.12 system efficiency is 0.36 for template matrix=2x2 while efficiency is 0.98 for template matrix=9x9. The significant improvement in system efficiency is solely due to greater amount of data locality since all other software and hardware parameters are kept constant.

As observed in Sections 7.1, 7.2, 7.3, and 7.4, as shared-bus-width widens, more bandwidth is available to serve a greater number of processors. Therefore, both shared-bus-width and data locality contribute to a better system efficiency. Thus, column shaded areas in tables are larger as sections advance.

Table 5.10 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

Execution Time vs. Number of Processors when *nl*=256				
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	0.9198	1.9697	7.6360	17.0774
2	0.4600	0.9849	3.8181	8.5387
4	0.2307	0.4932	1.9098	4.2701
8	0.1170	0.2482	0.9566	2.1369
16	0.0819	0.1277	0.4820	1.0723
32	0.0800	0.0864	0.2487	0.5442
64	0.0800	0.0812	0.1401	0.2885

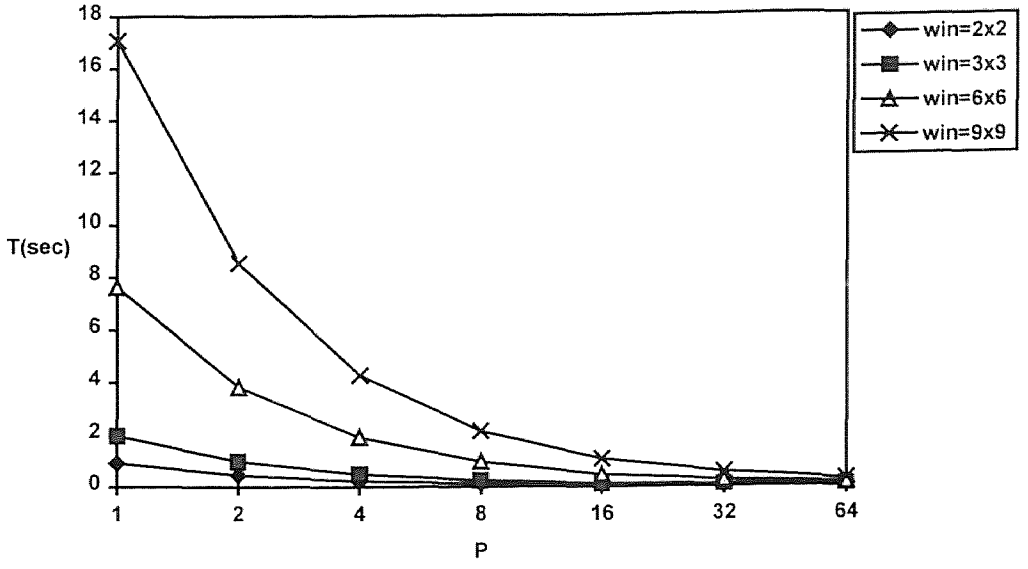

Figure 5.11 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

Table 5.11 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

Speedup vs. Number of Processors, *nl*=256				
P	S(2x2)	S(3x3)	S(6x6)	S(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	1.9996	1.9999	1.9999	2.0000
4	3.9870	3.9937	3.9983	3.9993
8	7.8615	7.9359	7.9824	7.9917
16	11.2308	15.4244	15.8423	15.9260
32	11.4975	22.7975	30.7037	31.3807
64	11.4975	24.2574	54.5039	59.1938

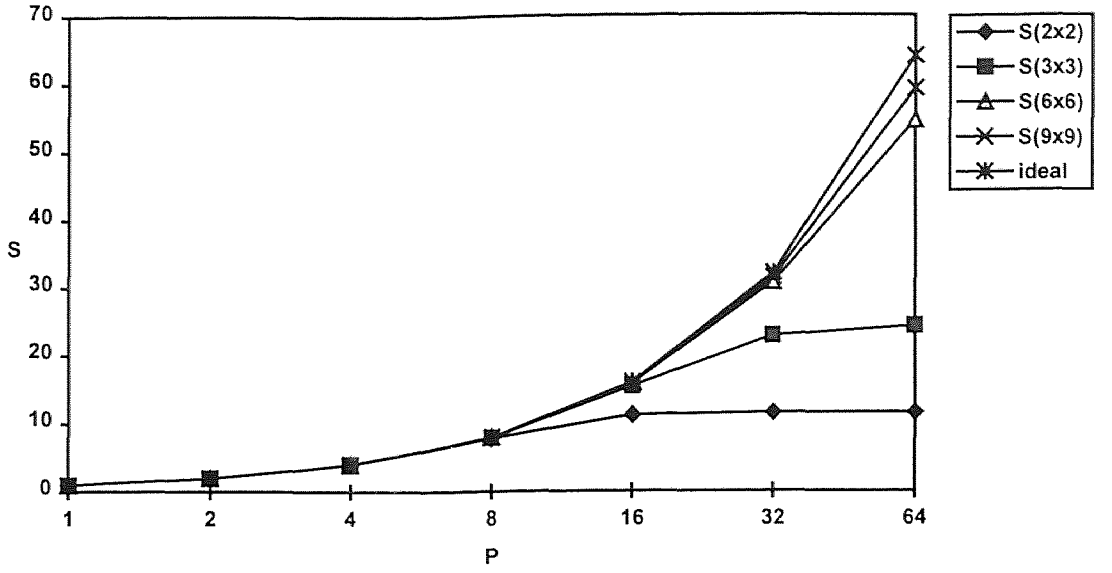

Figure 5.12 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

Table 5.12 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

Efficiency vs. Number of Processors, *nl*=256				
P	E(2x2)	E(3x3)	E(6x6)	E(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	0.9998	0.9999	1.0000	1.0000
4	0.9967	0.9984	0.9996	0.9998
8	0.9827	0.9920	0.9978	0.9990
16	0.7019	0.9640	0.9901	0.9954
32	0.3593	0.7124	0.9595	0.9806
64	0.1796	0.3790	0.8516	0.9249

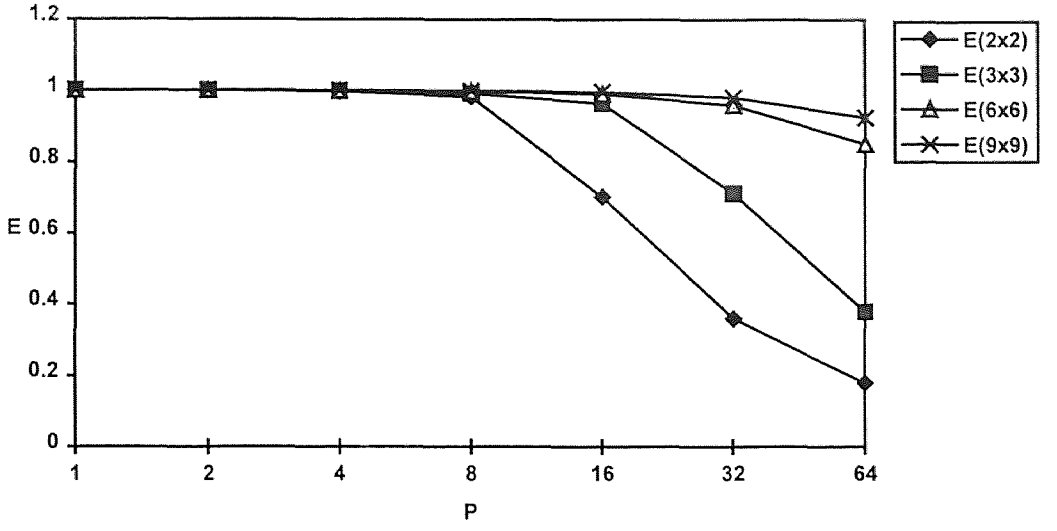

Figure 5.13 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=256

5.5 Effect of Spatial and Temporal Locality on a P-Node Twin-Prefetching Multiprocessor when Shared-Bus-Width is 512

Tables 5.13, 5.14 and 5.15, and figures 5.14, 5.15 and 5.16 show how the proposed twin-prefetching system performs as the size of *template matrix* (amount of data locality) varies, when shared-bus-width is 512. The shared-bus is wide enough to supply the necessary amount of bandwidth and effectively support 64 processors when template matrices 3x3, 6x6, and 9x9 are used. Therefore, there is little room to observe system performance improvement due to data locality. The comparison of last entries of the first two columns in Tables 5.13, 5.14 and 5.15 is the only possible demonstration of system performance, i.e., when a 2x2 template matrix (locality 4) is used, 32 processors

effectively execute the application while if a 3x3 template matrix (locality 81) is used the number of processors increases to 64. The significant improvement of system efficiency is solely due to greater amount of data locality in the latter application.

Table 5.13 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

Execution Time vs. Number of Processors when *nl*=512				
	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	0.8798	1.9290	7.5934	17.0328
2	0.4399	0.9645	3.7967	8.5164
4	0.2203	0.4826	1.8987	4.2586
8	0.1110	0.2421	0.9502	2.1302
16	0.0572	0.1229	0.4770	1.0670
32	0.0418	0.0651	0.2423	0.5376
64	0.0400	0.0457	0.1290	0.2770

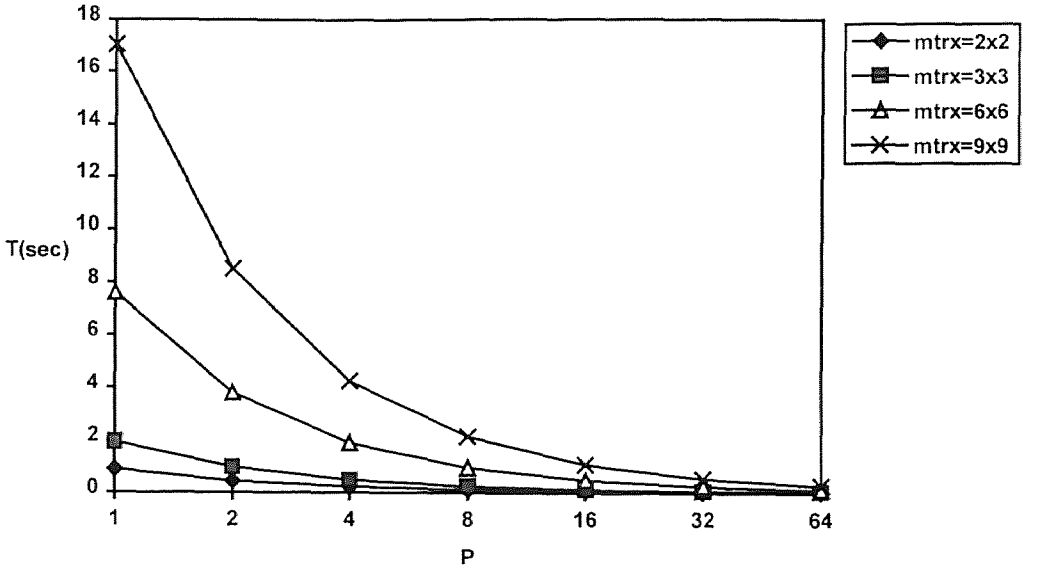

Figure 5.14 Execution Time vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

Table 5.14 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

Speedup *vs.* Number of Processors, *nl*=512				
P	S(2x2)	S(3x3)	S(6x6)	S(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	2.0000	2.0000	2.0000	2.0000
4	3.9936	3.9971	3.9993	3.9996
8	7.9261	7.9678	7.9914	7.9959
16	15.3811	15.6957	15.9191	15.9633
32	21.0478	29.6313	31.3388	31.6830
64	21.9950	42.2101	58.8636	61.4903

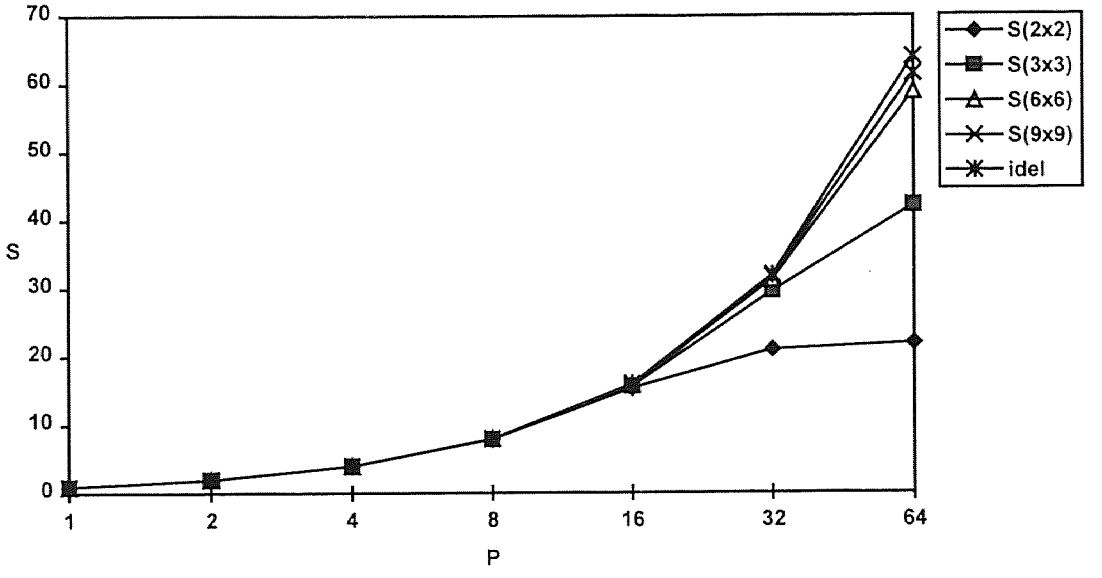

Figure 5.15 Speedup vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

Table 5.15 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

Efficiency vs. Number of Processors, *nl*=512				
P	E(2x2)	E(3x3)	E(6x6)	E(9x9)
1	1.0000	1.0000	1.0000	1.0000
2	1.0000	1.0000	1.0000	1.0000
4	0.9984	0.9993	0.9998	0.9999
8	0.9908	0.9960	0.9989	0.9995
16	0.9613	0.9810	0.9949	0.9977
32	0.6577	0.9260	0.9793	0.9901
64	0.3437	0.6595	0.9197	0.9608

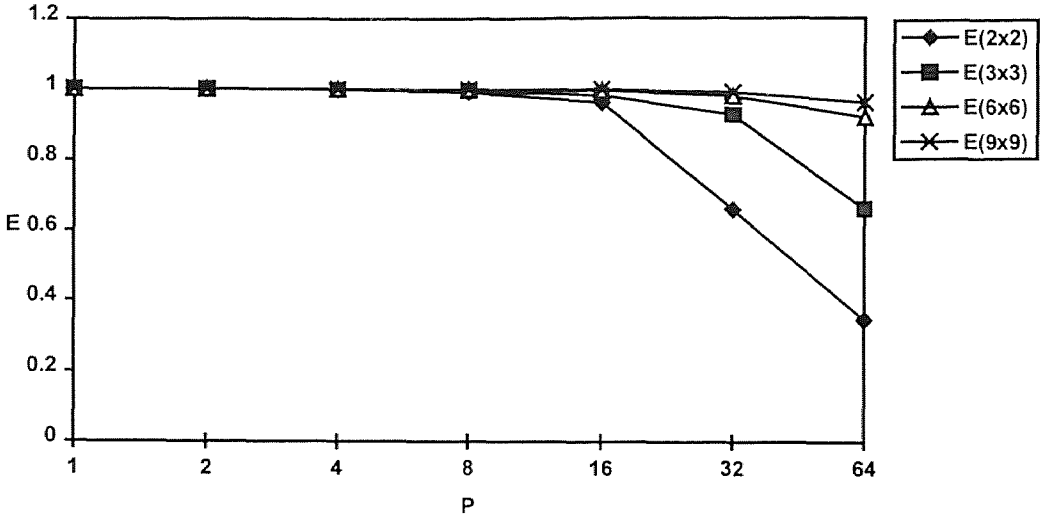

Figure 5.16 Efficiency vs. Number of Processors for *template matrix*=2x2, 3x3, 6x6, 9x9 when *nl*=512

5.6 Communication Overhead

In the convolution algorithm, if α output matrix rows are to be produced $\alpha+m_r-1$ rows have to be present (assuming partitioning by dividing total number of rows by P). Every processor in a message passing multicomputer system would have to transfer (communicate) the m_r-1 rows from other nodes. In a multiprocessor system every node retrieves from shared-memory an extra m_r-1 rows in order to produce the α output rows. Thus, a P-node parallel computer system would have to communicate at least $P(m_r-1)$ extra rows of data to convolute an image. The ratio $(m_r-1)/\alpha$ is a direct measure of the communication overhead and its value is expected to be small (much smaller than 1). We allow ratio $(m_r-1)/\alpha$ to take values much larger that one in order to test the performance of the proposed architecture with applications employing large amount of communication overhead. Table 5.16 consists of execution times when $\alpha=1$; thus, the ratio $(m_r-1)/\alpha$ becomes 1/1, 2/1, 5/1, and 8/1. Table 5.17 consists of execution times when $\alpha=2$; thus, the ratio $(m_r-1)/\alpha$ becomes 1/2, 2/2, 5/2, and 8/2. The difference between corresponding timing entries in Table 5.16 and 5.17 is small. Insignificant is also the difference of entries in Table 5.16 and 5.17 and the corresponding entries in Table 5.13 for which $\alpha=32$. This strongly indicates that the proposed system could handle applications with large amount of communication.

Table 5.16 Execution Time vs. Number of Processors when α=1 (*nl*=512)

	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	0.9079	1.9771	7.7040	17.2106
2	0.4543	0.9890	3.8524	8.6057
4	0.2272	0.4945	1.9262	4.3029
8	0.1136	0.2473	0.9632	2.1516
16	0.0605	0.1238	0.4818	1.0760
32	0.0605	0.0804	0.2413	0.5386
64	0.0604	0.0802	0.1406	0.2711

Table 5.17 Execution Time vs. Number of Processors when α=2 (*nl*=512)

	T(*sec*)	T(*sec*)	T(*sec*)	T(*sec*)
P	*tmpl.mtrx*=2x2	*tmpl.mtrx*=3x3	*tmpl.mtrx*=6x6	*tmpl.mtrx*=9x9
1	0.8933	1.9522	7.6471	17.1187
2	0.4469	0.9763	3.8238	8.5596
4	0.2235	0.4882	1.9120	4.2798
8	0.1118	0.2442	0.9561	2.1401
16	0.0561	0.1222	0.4783	1.0703
32	0.0499	0.0616	0.2396	0.5358
64	0.0499	0.0600	0.1212	0.2693

CHAPTER 6

DETERMINATION OF THE OPTIMAL SIZE OF PREFETCHING CACHES

In this chapter we investigate the effect of various prefetching cache sizes on the performance of all configurations of the *twin-prefetching* shared-memory multiprocessor system. The prefetching cache was varied (*ten* different values) from a few *Kbytes* to 4 *Mbytes* to determine the cost-performance optimal size. The twin-prefetching cache's function is to temporarily hold blocks of input and output data. Its size is an important design parameter, affecting performance, cost, and volume of the multiprocessor system.

Four different applications (Convolution utilizing *four* different template matrices) were executed on the 350 configurations of hardware parameters (varying *shared-bus-width*, number of processors and prefetching cache size). From the 1400 runs (Appendix C), a careful collection of results, shown in Tables 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 6.12, 6.13, and 6.14 enable the optimal prefetching cache size selection for every P-node based system.

An important observation, which assists the analysis of results and involves the availability of *shared-bus* bandwidth and categorization of applications, is described in Section 6.1. The reasoning behind the selection of results is explained in Section 6.2. Results are analyzed in Section 6.3 and presented in Sections 6.4-6.10. A discussion of results is given in Section 6.11.

The notation used to label variables in the tables is as follows: P stands for the number of processors, *nl* stands for the *shared-bus-width*, *tm.wn* stands for the size of the

template window, *cszvr* stands for the prefetching cache size, *csz* stands for the size of the smallest prefetching cache for which the best execution time occurs, L95% stands for the smallest cache size that gives at least 95% of the performance of *csz*, and *Time* stands for the execution time of the application.

Prefetching cache size is measured in *Kbytes* while execution time is measured in*seconds*. All execution times of applications are based on the Convolution of *eight* continuous images having resolution 1024x1024.

6.1 Shared Bus Contention Cases

Chapters 2 and 3 establish that all applications executed on the *twin-prefetching* shared memory multiprocessor fall into two *cases* of *shared-bus* bandwidth, depending on *Twratio*. When *Twratio* is greater or equal to P there is enough bus bandwidth for all processing elements, while bottleneck appears for smaller values of *Twratio*. More clearly *Case I* and *Case II* are stated as

I. *Twratio* $\geq$ P adequate *shared-bus* bandwidth for all processors

II. *Twratio* $<$ P bus contention - bottleneck

For a specific P-node based system and a specific application we test *ten* different values for the prefetching cache size. For *seven* different values of P, *five* different values of *nl*, and *four* applications, 140 sets of timings were recorded (Appendix C). After a detailed study of all sets of timings, it was observed that some sets of timings included a minimum value of time and some sets didn't. An example of no minimum occurrence is

shown in Table 6.1. Two examples of minimum occurrence are shown in Table 6.2 and Table 6.3.

Table 6.1 Prefetching cache size vs. time when P=8, *nl*=32, and *template window*=2x2

P=8, *Twratio*=1, *nl*=32, *template window*.=2x2										
cszvr	12	20	36	68	132	261	517	1029	2054	4102
Time	.9451	.7874	.7084	.6689	.6492	.6393	.6344	.6319	.6307	.6301

Table 6.2 Prefetching cache size vs. time when P=8, *nl*=128, and *template window*=2x2

P=8, *Twratio*=5, *nl*=128, *template window*.=2x2										
cszvr	12	20	36	68	132	261	517	1029	2054	4102
Time	.2374	.1974	.1776	.1679	.1635	.1621	.1631	.1670	.1757	.1758

Table 6.3 Prefetching cache size vs. time when P=8, *nl*=512, and *template window*=2x2

P=8, *Twratio*=20, *nl*=512, *template window*.=2x2										
cszvr	12	20	36	68	132	261	517	1029	2054	4102
Time	.113	.111	.1109	.11058	.11059	.1110	.1119	.1139	.117	.117

Further more it was observed that a minimum occurs when *Twratio* is greater or equal to P/2 and no minimum value occurs for *Twratio* less than P/2. In other words *Case I,* above, divides into two cases, and thus

I. *Twratio* < P/2 (great need of bus bandwidth-bottleneck)

II. P/2 ≥ *Twratio* > P (need of bus bandwidth-bottleneck)

III. *Twratio* ≥ P (bus bandwidth availability)

The importance of the three *cases* above is that they point out the amount the *shared-bus* contention. Thus, by looking at a table entry (Table 6.4 through Table 6.14) we observe not only the prefetching cache size needed for the particular hardware configuration but also how good the performance of the system is, i.e., the less the bus contention the better the execution time and the better the performance. Timings falling within *Case III* (*shared-bus* bandwidth availability for all processors) are accompanied by double quotes (") on the right side of their value. Timings falling within *Case II* (*shared-bus* bandwidth in need) are accompanied by single quotes (') on the right side of their value. Lastly, timings falling within *Case I* (*shared-bus* bandwidth in great need) are accompanied by no quotes on the right side of their value.

It should be noted that it is unknown whether larger values of *cszvr* would have produced a minimum timing in *Case I*. No higher values were tried because they are of no interest to our work for two reasons. One is because results show that a much smaller prefetching cache size provides performance very close to the maximum performance. The other reason is that very large sizes of prefetching cache would be inappropriate for our low cost system.

6.2 Selection of Results

We obtained 140 sets of ten timings each (Appendix C) running *four* applications on all variations (*seven* different values of P, *five* different values of *nl, ten* different values of prefetching cache) of the *twin-prefetching* P-node multiprocessor system. If a set provides a minimum timing value (best performance), the corresponding prefetching cache size is recorded in Tables 6.4 through 6.14 under the label *csz (Case I and Case II)*. Under the

label L95% we record the smallest prefetching cache size in a set, which provides at least 95% of the performance of *csz* (example results given in Table 6.2 and Table 6.3). If no minimum time occurs *(Case I)*, then no prefetching cache size value is recorded under the label *csz*. Since in this case the best time occurs when the largest prefetching cache size is tried (*4 Mbytes*) we record as the L95% value the smallest prefetching cache size that gives at least 95% of the performance achieved at *4 Mbytes* (example shown in Table 6.1). Observing timings in Table 6.1 and results in Appendix C, we conclude that the error of no minimum occurrence is virtually zero. This is due to the fact that timings which correspond to the larger prefetching cache sizes are very close to the timings which correspond to much smaller prefetching cache sizes.

Since our goal is to design a low cost DSP-based image-processing multiprocessor system, the smaller the size of the prefetching cache the better. Results point to a large gap between *csz* value and its corresponding L95% value. The difference in size between *csz* and L95% increases, as P becomes smaller. It should be noted, for the better understanding of results and more accurate evaluation of the system, that a difference of a few tens of *Kbytes* in the L95% prefetching cache size, when keeping *nl* constant and varying the *template window*, does not imply a bus bandwidth related prefetching cache size increase or decrease, but one that comes directly from the algorithm. More precisely, in order to produce k output rows or columns one has to load into the prefetching cache $k+c-1$ rows or columns, where c is the number of rows or columns of the *template window*. Thus, for the same number of output rows we need to load different number of input data rows depending on the size of the *template window*. This variation is small and negligible reaching a maximum value of 29 *Kbytes*. The

number 29 *Kbytes* could be calculated as follows. The largest *template window* used is 9x9 and the smallest 2x2. The respective extra rows loaded are *eight* and *one*. Their difference is *seven*. Every row of the input image is 1024 pixels. Since pixels are store as 32-bit elements (32-bit DSP-four bytes each pixel) in the prefetching cache the maximum of *seven* extra rows would take 4x1024x7=28 *Kbytes* of memory space. There is *one Kbyte* of difference between the observed and calculated maximum variation. The additional one is the round up *Kbyte* from the wrap around operation of column pixels. Therefore, 28+1=29. This difference could be seen in some column results of Tables 6.8, 6.9, 6.10, and 6.11 where the minimum value is 12 *Kbytes* and the maximum 41 *Kbytes*.

6.3 Analysis of Results

In Tables 6.4, 6.5, 6.6, and 6.7 we observe the changes of *csz* and L95%, as we vary P and *nl*, when *template window* is 2x2, 3x3, 6x6, and 9x9, respectively. In Tables 6.8, 6.9, 6.10, 6.11, 6.12, 6.13, and 6.14 we observe the changes of *csz* and L95%, for a particular P-node *twin-prefetching* multiprocessor for all changes of the *template window* and *nl*.

Examining results in Tables 6.4, 6.5, 6.6, and 6.7 we observe the need for less prefetching cache (L95%) as *nl* increases. There is also a greater need of prefetching cache (L95%) as the number of processors (P) increases. It is also evident that the *csz* and the L95% values, approach each other (much faster from the *csz* side) for higher values of *nl* and P. We also observe the need for a larger prefetching cache size (L95%) when moving from one category of bus bandwidth to another, i.e. from double quotes to single to no quotes. These observations are to be considered when designing a *twin-prefetching*

multiprocessor. It should be mentioned that the emphasis on the L95% values in our analysis is due to cost attributes.

The most significant observation (Tables 6.4-6.14) *is that L95% values, for all changes of hardware or software parameters, fall within the range of 12 to 290 Kbytes.* In other words, any *twin-prefetching* multiprocessor system tested could obtain almost its maximum performance using only a small prefetching cache.

Let us now focus on Tables 6.8, 6.9, 6.10, 6.11, 6.12, 6.13 and 6.14 where we concentrate specifically on the cost-performance optimal prefetching cache size of a particular P-node *twin-prefetching* multiprocessor. For every specific value of P and *nl* (different shared-bus availability) four applications are executed and four different prefetching cache sizes (L95%) are utilized. We select as optimal prefetching cache size the largest of the four sizes. In other words, we select the prefetching cache size that covers all applications executed on the system. It is important to note that we consider optimal value the cost-performance optimal prefetching cache size value (*L95%*).

6.4 Optimal Selection of Prefetching Cache Size when P=1

When P=1 (Table 6.8) *csz* is in the range of *Mbytes* while the L95% value is always less than 100 *Kbytes*. Respective L95% values for nl=32, 64, 128, 256, 512 are 65, 49, 41, 41, 41 *Kbytes*. The largest L95% prefetching cache size value (65 *Kbytes*) occurs when *nl*=32 and *template window*=9x9, and the smallest prefetching cache size (12 *Kbytes*) occurs when *nl*=256 or *nl*=512 and *template window*=2x2. To be more precise, when *nl*=32 and *template window*=9x9, 65*Kbytes* of prefetching cache provide 96.5% of the maximum performance while when *nl*=256 and *template window*=2x2 the prefetching cache size

provides 99% of the maximum performance (timings for calculations of L95% values are taken from Appendix C). The above example L95% values are presented in order to demonstrate how close to the best performance is the at-least-95%-of-the-best-performance.

Comparing Table 6.8 with Tables 6.9-6.14 we observe that the difference between *csz* and L95% is greater for smaller values of P. We notice that for all changing values of *nl* and *template window*, there is enough bus bandwidth (double quotes next to every value in the Table 6.8). (It is a little strange talking about bus bandwidth availability when P=1 but let us be reminded that there are two twin controllers utilizing the bus.)

We could finally say that a minimum size value of 65 *Kbytes* (a small memory size in today's technology) covers all hardware and software requirements when P=1.

6.5 Optimal Selection of Prefetching Cache Size when P=2

When P=2 (Table 6.9) *csz* is in the range of *Mbytes* (except when the *template window=2x2* where *csz* is half a *Mbyte)* while L95% values are always less than 100 *Kbytes*. Respective L95% values for nl=32, 64, 128, 256, 512 are 65, 49, 41, 41, 41 *Kbytes*. The largest L95% prefetching cache size value (65 *Kbytes*) occurs when *nl*=32 and *template window*=9x9, and the smallest prefetching cache size (12 *Kbytes*) occurs when *nl*=256 or *nl*=512 and *template window*=2x2.

We notice that for all changing values of *nl* and *template window*, there is enough bus bandwidth (double quotes next to almost every value in the Table 6.9).

Finally, we could say that a minimum size value of 65 *Kbytes* covers all hardware and software requirements when P=2.

6.6 Optimal Selection of Prefetching Cache Size when P=4

For P=4 (Table 6.10) *csz* takes values in the range 133*Kbytes*≤*csz*≤*4Mbytes* (*4Mbytes* are used as the upper limit of *csz* because there is one case of no minimum convergence in the Table) while L95% values are all less than 137 *Kbytes*. For nl=32, 64, 128, 256, 512, L95% values are 137, 68, 41, 41, 41 *Kbytes* respectively. The largest L95% prefetching cache size value (137 *Kbytes*) occurs when *nl*=32 and *template window*=3x3, and the smallest prefetching cache size (12 *Kbytes*) occurs when *nl*=256 or *nl*=512 and *template window*=2x2.

When *nl*=32 and *template window*=2x2, the application falls in *Case I* (no quotes-great need of bus bandwidth). For *nl*=32 & *template window*=3x3 and *nl*=64 & *template window*=2x2 , the application falls in *Case II* (single quotes - need of bus bandwidth. All other hardware and software combinations fall into *Case III.*

We could finally say that when P=4, a minimum prefetching cache size value of 137 *Kbytes* covers all hardware and software requirements.

6.7 Optimal Selection of Prefetching Cache Size when P=8

Table 6.11 (P=8) provides *csz* values in the range 68*Kbytes*≤*csz*≤*4Mbytes* (*4Mbytes* are used as the upper limit of *csz* because there is at least one case of no minimum convergence in Table 6.11) while all L95% sizes receive values less than 137 *Kbytes*. For nl=32, 64, 128, 256, 512, L95% values are 137, 137, 68, 41, 41 *Kbytes* respectively. The largest L95% prefetching cache size value (137 *Kbytes*) occurs when *nl*=32 and *template*

window=3x3, and the smallest prefetching cache size (12 *Kbytes*) occurs when *nl*=256 or *nl*=512 and *template window*=2x2.

For nl=32, 64 and template window=2x2, 3x3 we observe some shortage of *shared-bus* bandwidth while all other Table entries fall in Case III (shared-bus bandwidth availability).

We could finally say that when P=8, a minimum prefetching cache size value of 137 *Kbytes* covers all hardware and software requirements.

6.8 Optimal Selection of Prefetching Cache Size when P=16

Table 6.12 (P=16) provides *csz* values in the range $36Kbytes \leq csz \leq 4Mbytes$ (*4Mbytes* are used as the upper limit of *csz* because there is at least one case of no minimum convergence in Table 6.11) while all L95% sizes receive values less than 149 *Kbytes*. For nl=32, 64, 128, 256, 512, L95% values are 149, 137, 133, 68, 41 *Kbytes* respectively. The largest L95% prefetching cache size value (149 *Kbytes*) occurs when *nl*=32 and *template window*=6x6, and the smallest prefetching cache size (16 *Kbytes*) occurs when *nl*=512 and *template window*=3x3.

For nl=32, 64, 128 and template window=2x2, 3x3, 6x6 we observe some shortage of *shared-bus* bandwidth while all other Table entries fall in Case III (shared-bus bandwidth availability).

We could finally say that when P=16, a minimum prefetching cache size value of 149 *Kbytes* covers all hardware and software requirements.

6.9 Optimal Selection of Prefetching Cache Size when P=32

Table 6.13 (P=32) provides *csz* values in the range 40*Kbytes*≤*csz*≤*4Mbytes* (*4Mbytes* are used as the upper limit of *csz* because there is at least one case of no minimum convergence in Table 6.11) while all L95% sizes receive values less than 277 *Kbytes*. For nl=32, 64, 128, 256, 512 L95% values are 277, 149, 137, 133, 68 *Kbytes* respectively. The largest L95% prefetching cache size value (277 *Kbytes*) occurs when *nl*=32 and *template window*=6x6, and the smallest prefetching cache size (24 *Kbytes*) occurs when *nl*=512 and *template window*=3x3.

A severe shortage of *shared-bus* bandwidth is observed when *nl*=32, 64, and a milder shortage when *nl*=128, 256. Still, *nl*=512 is able to effectively support all applications executed on the system.

We could finally say that when P=32, a minimum prefetching cache size value of 277 *Kbytes* covers all hardware and software requirements.

6.10 Optimal Selection of Prefetching Cache Size when P=64

Table 6.14 (P=64) provides *csz* values in the range 65*Kbytes*≤*csz*≤*4Mbytes* (*4Mbytes* are used as the upper limit of *csz* because there is at least one case of no minimum convergence in Table 6.11) while all L95% sizes receive values less than 290 *Kbytes*. For nl=32, 64, 128, 256, 512 L95% values are 290, 290, 277, 149, 133 *Kbytes* respectively. The largest L95% prefetching cache size value (290*Kbytes*) occurs when *nl*=32 and *template window*=6x6, and the smallest prefetching cache size (41 *Kbytes*) occurs when *nl*=512 and *template window*=9x9.

A severe shortage of *shared-bus* bandwidth is observed when *nl*=32, 64, 128 and a milder shortage when *nl*=256. Still, *nl*=512 is able to effectively support applications with template window 3x3, 6x6 and 9x9.

We could finally say that when P=64, a minimum prefetching cache size value of 290 *Kbytes* covers all hardware and software requirements.

6.11 Discussion of Results

Results show that by utilizing a small prefetching cache size (12 to 290 *Kbytes*) we achieve at least 95% of maximum performance of any P-based *twin-prefetching* multiprocessor (P=1, 2, 4, 8, 16, 32, 64). Despite its low cost, this small amount of prefetching cache size along with *twin-prefetching* mechanism and a wider *shared-bus* makes possible the 100% utilization of all processors' bandwidth.

Furthermore, due to the range $12Kbytes \leq L95\% \leq 290Kbytes$, we conclude that the prefetching cache size is relatively insensitive to the amount of temporal and spatial locality embedded in different applications. For the same reason the prefetching cache size is relatively insensitive to *shared-bus* bandwidth and number of processors. The insensitivity of the prefetching cache to the above parameters is extremely important for the hardware implementation of a *twin-prefetching* system.

Table 6.15 shows the sizes of prefetching cache which should be utilized in the implementation of a *twin-prefetching* multiprocessor with number of processors P and *shared-bus* width *nl*. Every entry in Table 6.15 is the L95% value, which allows at least 95% of the best performance of all applications executed.

290 *Kbytes* (maximum amount of prefetching cache in Table 6.15) is a small quantity of fast memory in today's advanced technology. Chapter 4 suggests that configurations (P and *nl*) of *twin-prefetching* multiprocessor systems shown in Table 6.15, employing *twin-prefetching* caches greater than 150 *Kbytes* should not be implemented because of a lack of *shared-bus* bandwidth availability. Chapter 4 also suggest that configurations of *twin-prefetching* multiprocessors summarized in Table 6.15, employing less than 100 *Kbytes,* are definitely worth building. It is our opinion that an implementation of a *twin-prefetching* multiprocessors should carry double or triple the amount shown in Table 6.15. The justification is that a small quantity, additional to the value shown in Table 6.15, would cost very little and cover any unexpected application needs.

All P-node *twin-prefetching* multiprocessors systems were tested with large, high-resolution (*eight* 1024x1024) images. Due to the nature of *twin-prefetching* mechanism, the real-time execution of even a larger image would not require a larger prefetching cache.

Table 6.4 Optimal prefetching cache size when *template window*=2x2

tm.wn 2x2	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
P	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
1	1029"	36"	1029"	36"	1029"	20"	1029"	12"	1029"	12"
2	517'	36'	517"	36"	516"	20"	516"	12"	516"	12"
4		133	260'	68'	133"	20"	133"	12"	133"	12"
8		133		133	161'	68'	68"	12"	68"	12"
16		133		133		133	260'	68'	36"	20"
32		133		133		133		133	133'	68'
64		133		133		133		133		133

Table 6.5 Optimal prefetching cache size when *template window*=3x3

tm.wn 3x3	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
P	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
1	1033"	73"	1033"	40"	1033"	24"	1033"	16"	1033"	16"
2	1033"	73"	1033"	40"	1033"	24"	1033"	16"	1033"	16"
4	521'	137'	265"	40"	265"	24"	265"	16"	265"	16"
8		137	265'	137'	137"	24"	248"	16"	137"	16"
16		137		137	265'	73'	73"	24"	73"	16"
32		137		137		137	137'	73'	40"	24"
64		137		137		137		137	137'	73'

Table 6.6 Optimal prefetching cache size when *template window*=6x6

tm.wn 6x6	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
P	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
1	2074"	53"	2073"	37"	2073"	29"	2073"	29"	2073"	29"
2	1047"	53"	1047"	37"	1047"	29"	1047"	29"	1047"	29"
4	534"	53"	534"	37"	534"	29"	534"	29"	534"	29"
8	149"	53"	149"	37"	149"	29"	149"	29"	149"	29"
16	534'	149"	85"	53'	85"	37"	85"	29"	85"	29"
32		277	364'	149'	85"	53'	53"	37"	53"	29"
64		277		277		277	534'	149'	85"	53"

Table 6.7 Optimal prefetching cache size when *template window*=9x9

tm.wn 9x9	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
P	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
1	2089"	65"	2089"	49"	2089"	41"	2089"	41"	2089"	41"
2	2089"	65"	2089"	49"	2089"	41"	2089"	41"	2089"	41"
4	290"	65"	547"	49"	547"	41"	547"	41"	547"	41"
8	290"	97"	290"	49"	290"	41"	290"	41"	290"	41"
16	161"	97"	161"	49"	161"	41"	161"	41"	161"	41"
32	1061'	161'	97"	49'	97"	49"	97"	41"	97"	41"
64		290	547'	290'	97"	97"	65"	49"	65"	49"

Table 6.8 Optimal prefetching cache size when P=1

P=1	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2	1029"	36"	1029"	36"	1029"	20"	1029"	12"	1029"	12"
3x3	1033"	40"	1033"	40"	1033"	24"	1033"	16"	1033"	16"
6x6	2074"	53"	2073"	37"	2073"	29"	2073"	29"	2073"	29"
9x9	2089"	65"	2089"	49"	2089"	41"	2089"	41"	2089"	41"

Table 6.9 Optimal prefetching cache size when P=2

P=2	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2	517'	36'	517"	36"	517"	20"	517"	12"	517"	12"
3x3	1033"	40"	1033"	40"	1033"	24"	1033"	16"	1033"	16"
6x6	1047"	53"	1047"	37"	1047"	29"	1047"	29"	1047"	29"
9x9	2089"	65"	2089"	49"	2089"	41"	2089"	41"	2089"	41"

Table 6.10 Optimal prefetching cache size when P=4

P=4	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2		133	260'	68'	133"	20"	133"	12"	133"	12"
3x3	521'	137'	265"	40"	265"	24"	265"	16"	265"	16"
6x6	534"	53"	534"	37"	534"	29"	534"	29"	534"	29"
9x9	290"	65"	547"	49"	547"	41"	547"	41"	547"	41"

Table 6.11 Optimal prefetching cache size when P=8

P=8	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2		133		133	161'	68'	68"	12"	68"	12"
3x3		137	265'	137'	137"	24"	248"	16"	137"	16"
6x6	149"	53"	149"	37"	149"	29"	149"	29"	149"	29"
9x9	290"	97"	290"	49"	290"	41"	290"	41"	290"	41"

Table 6.12 Optimal prefetching cache size when P=16

P=16	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2		133		133		133	260'	68'	36"	20"
3x3		137		137	265'	73'	73"	24"	73"	16"
6x6	534'	149"	85"	53'	85"	37"	85"	29"	85"	29"
9x9	161"	97"	161"	49"	161"	41"	161"	41"	161"	41"

Table 6.13 Optimal prefetching cache size when P=32

P=32	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2		133		133		133		133	133'	68'
3x3		137		137		137	137'	73'	40"	24"
6x6		277	364'	149'	85"	53'	53"	37"	53"	29"
9x9	1061'	161'	97"	49'	97"	49"	97"	41"	97"	41"

Table 6.14 Optimal prefetching cache size when P=64

P=64	*nl*=32		*nl*=64		*nl*=128		*nl*=256		*nl*=512	
tm.wn	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%	*csz*	L95%
2x2		133		133		133		133		133
3x3		137		137		137		137	137'	73'
6x6		277		277		277	534'	149'	85"	53"
9x9		290	547'	290'	97"	97"	65"	49"	65"	41"

Table 6.15 Optimal prefetching cache size

P	*nl*=32	*nl*=64	*nl*=128	*nl*=256	*nl*=512
1	65	49	41	41	41
2	65	49	41	41	41
4	137	68	41	41	41
8	137	137	68	41	41
16	149	137	133	68	41
32	277	149	137	133	68
64	290	290	277	149	133

CHAPTER 7

CONCLUSIONS

This dissertation introduces, investigates, and evaluates, through simulation, a low-cost high-speed twin-prefetching DSP-based shared-bus shared-memory multiprocessor system for real-time image processing applications. The proposed architecture can effectively support 32 modern high-performance DSPs (ADSP-21060) transcending promising signals to the area of shared-memory multiprocessors and real-time image-processing. It should be noted that the number of effectively supported processors (32) is conservatively selected and it is based on the worst case scenario when the application with the smallest number of local neighborhood operations (2D convolution with template matrix 2x2) is executed. Applications employing high number of local neighborhood operations (e.g., 2D convolution with template matrix 9x9) achieve a system efficiency greater than 90% even when 64 processors share the bus. The number of neighborhood operations is a direct measure of the amount of data locality embedded in a particular application and the above conclusions demonstrate the significant effect of data locality on the system's performance. The positive response of both kinds of applications on the proposed multiprocessor system demonstrates its potential to meet, in a cost effective manner, the challenges not only of real-time digital image processing but also of other computationally intensive applications.

The cost of the proposed system is kept low because of the following reasons: (1) single bus-based systems are the least costly; (2) DSPs cost considerably less that RISC

or CISC microprocessors; and (3) as this work concludes, the amount of fast (expensive) memory needed for every prefetching cache is small (16 to 300 Kbytes).

In contrast to existing DSP-based multiprocessors, the proposed system sustains peak performance regardless of image size due to the twin-prefetching mechanism. It should be noted that in the applications run on recent DSP-based uniprocessor or multiprocessor systems the majority of instructions are multifunction, each one including a reference to data memory (where the image resides). It is necessary for the entire section of the assigned image to be stored on fast memory (SRAM) in order to guarantee single-cycle execution of all (multifunction) instructions; a basic approach for real-time DSP-based systems. However, it is quite expensive storing one or a series of input and output high resolution images on SRAM. It should be noted that even if the entire shared memory consisted of SRAM in a conventional shared-bus multiprocessor system (processor and bus having the same data width), still there would not be enough bandwidth to support more than a handful of processors. Therefore, the twin-prefetching caches placed between the processor and the shared bus serve two goals that benefit both cost and performance. One is the feasibility of utilizing a wider bus which provides the additional bandwidth required to effectively support a greater number of processors. The other is a significant cost reduction since two inexpensive small-size fast memories (prefetching caches) achieve the performance of a much larger and more expensive SRAM. It should be noted that, in the proposed system, the assigned to each processor image sections, regardless of being large or small are partitioned into smaller segments which are interchangeably loaded on the Twin1 and Twin2 prefetching caches.

Eight consecutive high-resolution images of size 1024x1024 were mainly used to investigate the processing power of the proposed system architecture. Our performance analysis has demonstrated the real-time potential of this system. This is of paramount importance in areas like digital video processing or robotic vision where processing sequences of 25-30 images per second is a common place. Below we provide some typical results of processing times that different configurations of our system can achieve in the case of convolution of 30 consecutive images (1024x1024) when a 3x3 template matrix is used. When the number of processors P=64 and shared bus width nl=512 execution time T=0.172 seconds; when P=32 and nl=512 T=0.245 seconds; when P=16 and nl=256 T=0..480 seconds; when P=8 and nl=256 T=0.940 seconds.

Further research will be contacted to evaluate the performance of the proposed multiprocessor system with additional digital image processing algorithms and neural network simulation for pattern recognition and classification. A NUMA shared-memory parallel computer formed by several clusters of the proposed system will be also investigated for applications requiring even greater computing power.

APPENDIX A

ASSEMBLER CODE FOR 2D CONVOLUTION

Example of assembler code listing for two-dimensional Convolution of a segment of an image (Mr x Nc) stored in prefetching cache, when *template window*=3x3:

```
.SEGMENT/ DMDATA dmsegment;
.VAR                                    inseg [(Mr+2)*(Nc+2)];
.VAR                                    outseg [Mr*Nc];
.ENDSEG;

.SEGMENT/PMDATA pmdataseg;
.VAR                                    template [3*3] = "tmpl.dat";
.ENDSEG;


.SEGMENT/PMCODE pmcodeseg;
setup:          M0=1;
                M1=Nc;
                M2= -(2*Nc+5) ;
                M8=1;
                M9=2 ;
                B0=inseg;               L0=0 ;
                B8=template ;           L8=@template ;
                B9=outseg + Mr * Nc ;   L9=@outseg ;
                F5=PM (template + 2) ;  /* store in register file to save cycle */

conv : F0=DM ( I0, M0 ) , F4=PM ( I8 ,M8 ) ;
                LCNTR=Mr , D0 in_row UNTIL LCE;
                  LCNTR=Nc ,        D0 in_col . UNTIL LCE;
                        F8=F0*F4, F8=F8+F12,   F0=DM (I0,M0), F4=PM (I8,M9);
                        F12=F0*F4,             F0=DM (I0,M1), PM (I9,M8)=F8;
                        F12=F0*F5, F8=F8+F12, F0=DM (I0,M0), F4=PM (I8,M8);
                        F12=F0*F4, F8=F8+F12, F0=DM (I0,M0), F4=PM (I8,M8);
                        F12=F0*F4, F8=F8+F12, F0=DM (I0,M1), F4=PM (I8,M8);
                        F12=F0*F4, F8=F8+F12, F0=DM (I0,M0), F4=PM (I8,M8);
                        F12=F0*F4, F8=F8+F12, F0=DM (I0,M0), F4=PM (I8,M8);
                        F12=F0*F4, F8=F8+F12, F0=DM (I0,M2), F4=PM (I8,M8);
in_col:                 F12=F0*F4, F8=F8+F12, F0=DM (I0,M0), F4=PM (I8,M8);
```

```
                MODIFY (I0,M0 ) ;
in_row:                 F0=DM (I0,M0) ;
                RTS (D), F8=F8+F12;
                PM (I9,M8)=F8;
                NOP;
ENDSEG;
```

/* The third coefficient is stored in the register file to free up a cycle to output a result. The first write to the output buffer is unused , the pointer then wraps around to the proper location at the start of output. */

APPENDIX B

EXECUTION TIME & SPEEDUP VS. SHARED-BUS-WIDTH

Template matrix =2x2

Table B.1 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

Execution Time vs. Shared-Bus-Width, *template matrix*=2x2							
nl	TP1(s)	TP2(s)	TP4(s)	TP8(s)	TP16(s)	TP32(s)	TP64(s)
32	1.4803	0.7413	0.6393	0.6393	0.6393	0.6393	0.6393
64	1.1600	0.5803	0.3226	0.3197	0.3197	0.3197	0.3197
128	0.9999	0.5001	0.2514	0.1621	0.1599	0.1599	0.1599
256	0.9198	0.4600	0.2307	0.1170	0.0819	0.0800	0.0800
512	0.8798	0.4399	0.2203	0.1110	0.0572	0.0418	0.0400

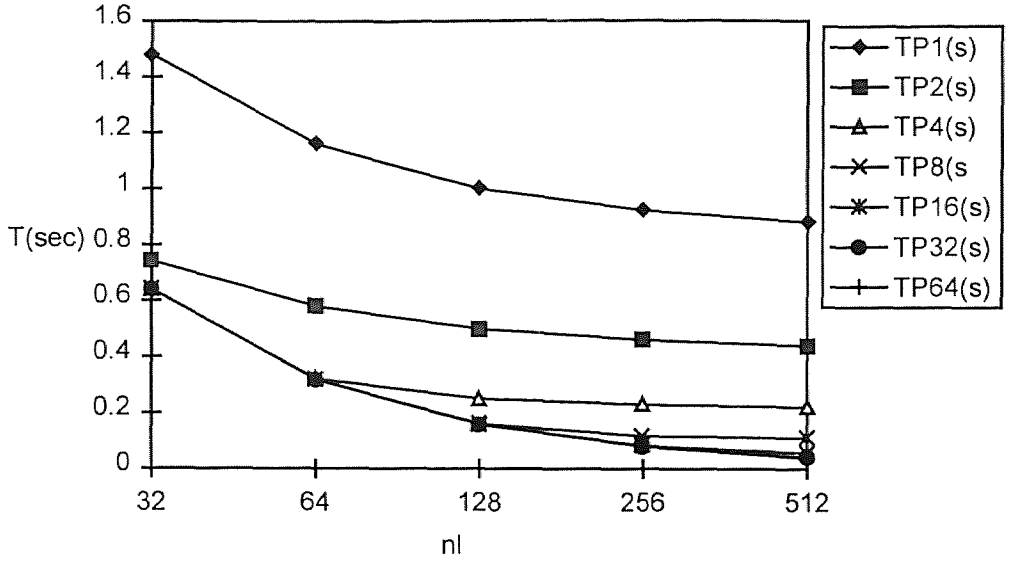

Figure B.1 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

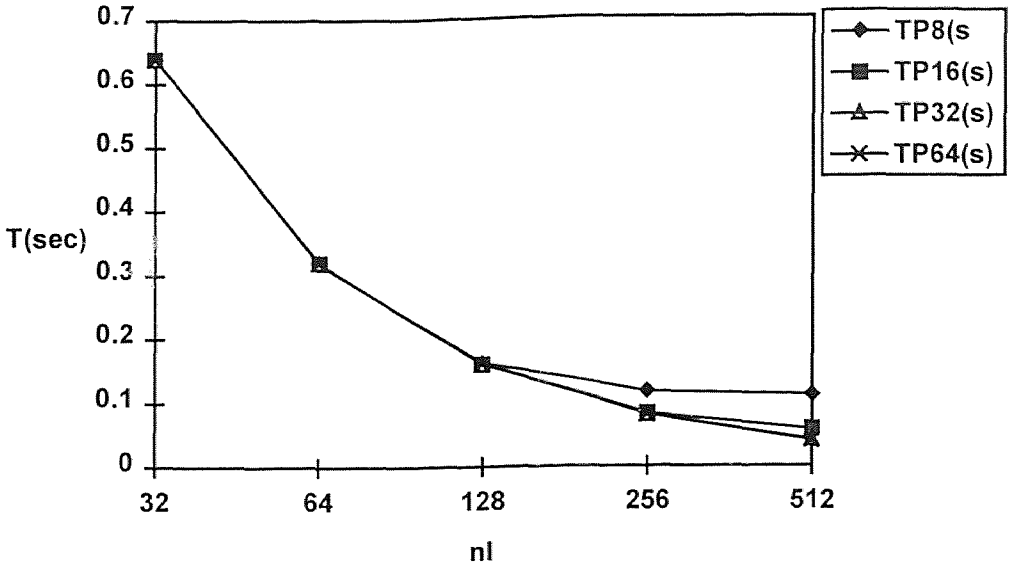

Figure B.2 Execution time vs. Shared-Bus-Width (*nl*) for P=8, 16, 32 and 64 when *template matrix*=2x2

Table B.2 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

Speedup vs. Shared-Bus-Width, *template matrix*=2x2							
nl	SP1	SP2	SP4	SP8(s	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.2759	1.2776	1.9845	2.0031	1.9969	1.9969	1.9969
128	1.4800	1.4820	2.5458	3.9444	3.9938	3.9938	3.9938
256	1.6087	1.6109	2.7662	5.5086	7.7927	7.9875	7.9875
512	1.6818	1.6841	2.9045	5.7568	11.2105	15.5854	15.9750

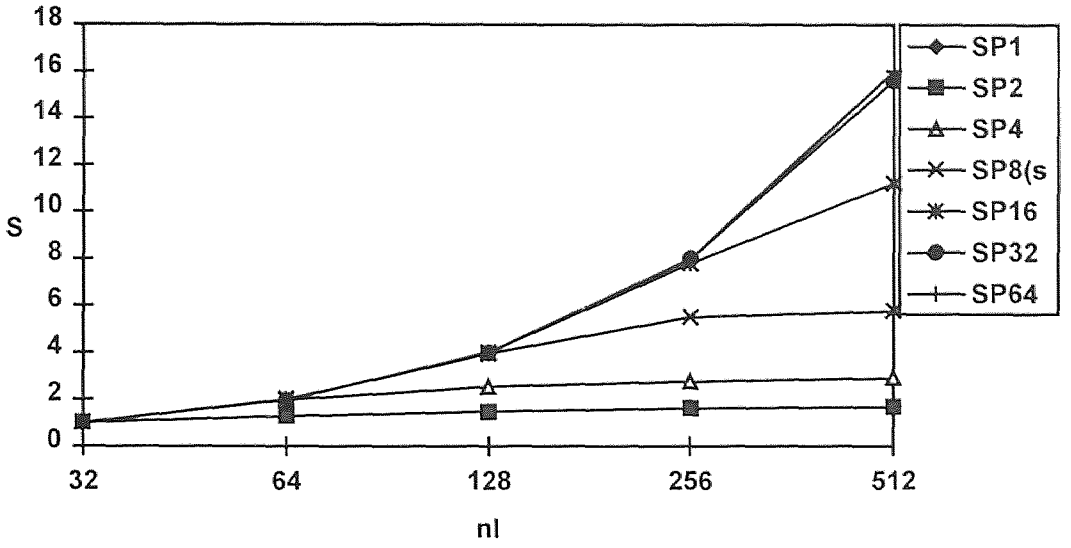

Figure B.3 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=2x2

Template matrix =3x3

Table B.3 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=3x3

Execution Time vs. Shared-Bus-Width, *template matrix*=3x3							
nl	TP1(s)	TP2(s)	TP4(s)	TP8(s	TP16(s)	TP32(s)	TP64(s)
32	2.5391	1.2701	0.6569	0.6495	0.6490	0.6490	0.6495
64	2.2137	1.1071	0.5564	0.3309	0.3248	0.3248	0.3243
128	2.0510	1.0251	0.5142	0.2604	0.1679	0.1624	0.1624
256	1.9697	0.9849	0.4932	0.2482	0.1277	0.0864	0.0812
512	1.9290	0.9645	0.4836	0.2421	0.1229	0.0651	0.0457

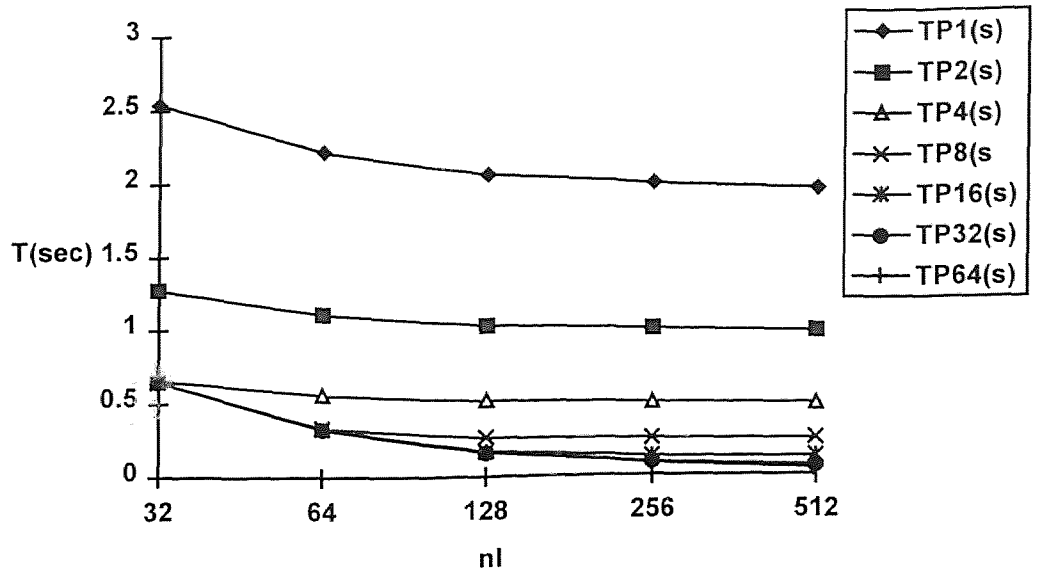

Figure B.4 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=3x3

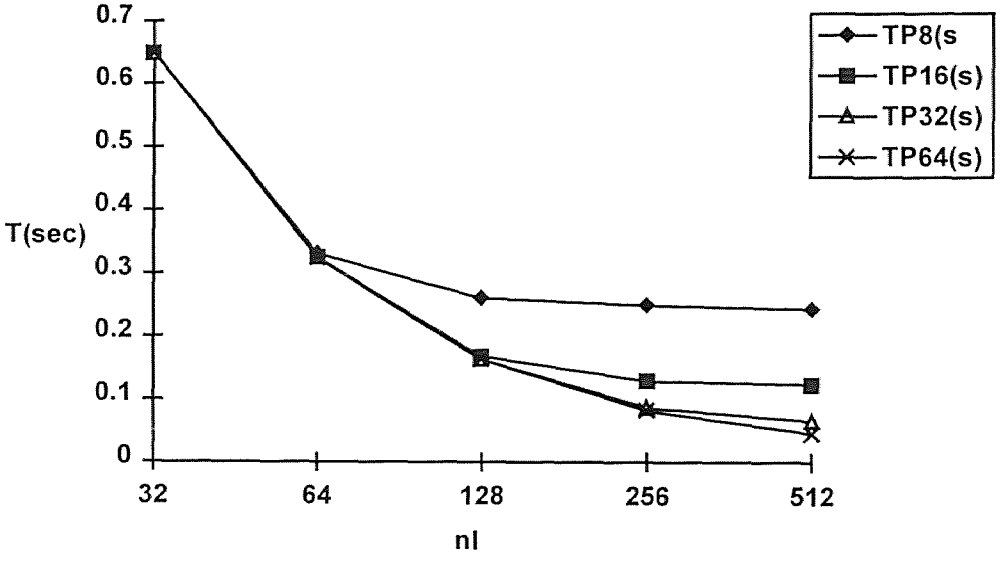

Figure B.5 Execution time vs. Shared-Bus-Width (*nl*) for P=8, 16, 32 and 64 when *template matrix*=3x3

Table B.4 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=3x3

Speedup vs. Shared-Bus-Width, *template matrix*=3x3							
nl	SP1	SP2	SP4	SP8(s	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.1470	1.1472	1.1806	1.9628	1.9982	1.9982	2.0028
128	1.2380	1.2390	1.2775	2.4942	3.8654	3.9963	3.9994
256	1.2891	1.2896	1.3319	2.6168	5.0822	7.5116	7.9988
512	1.3163	1.3168	1.3584	2.6828	5.2807	9.9693	14.2123

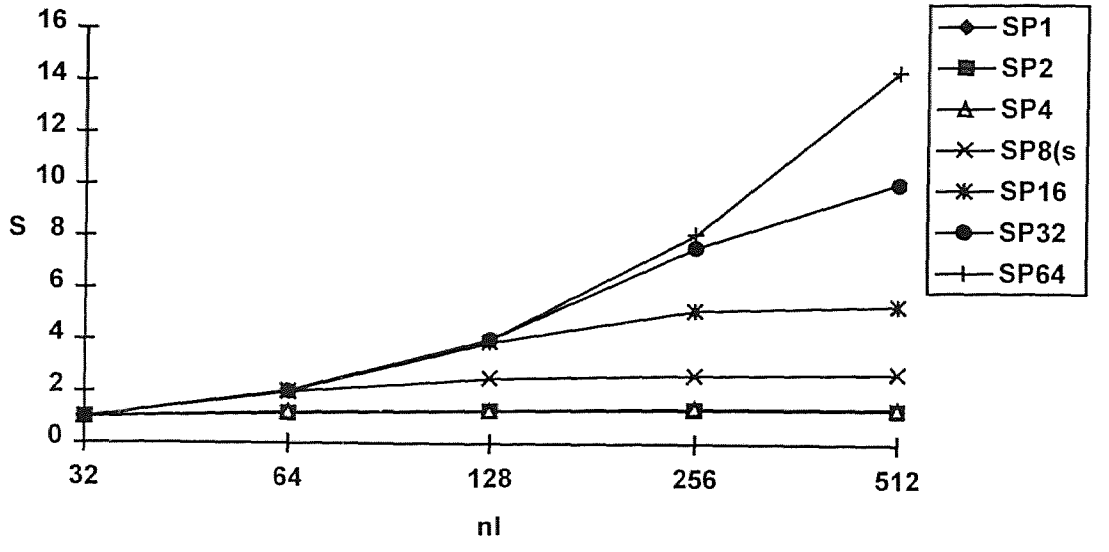

Figure B.6 Speedup vs. Shared-Bus-Width (*nl*) for P=8, 16, 32 and 64 when *template matrix*=3x3

Template matrix =6x6

Table B.5 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

Execution Time vs. Shared-Bus-Width, *template matrix*=6x6							
nl	TP1(s)	TP2(s)	TP4(s)	TP8(s	TP16(s)	TP32(s)	TP64(s)
32	8.232	4.117	2.064	1.046	0.702	0.680	0.680
64	7.892	3.946	1.976	0.998	0.512	0.360	0.340
128	7.721	3.860	1.932	0.969	0.492	0.261	0.190
256	7.636	3.818	1.910	0.957	0.482	0.248	0.140
512	7.593	3.797	1.899	0.950	0.476	0.242	0.129

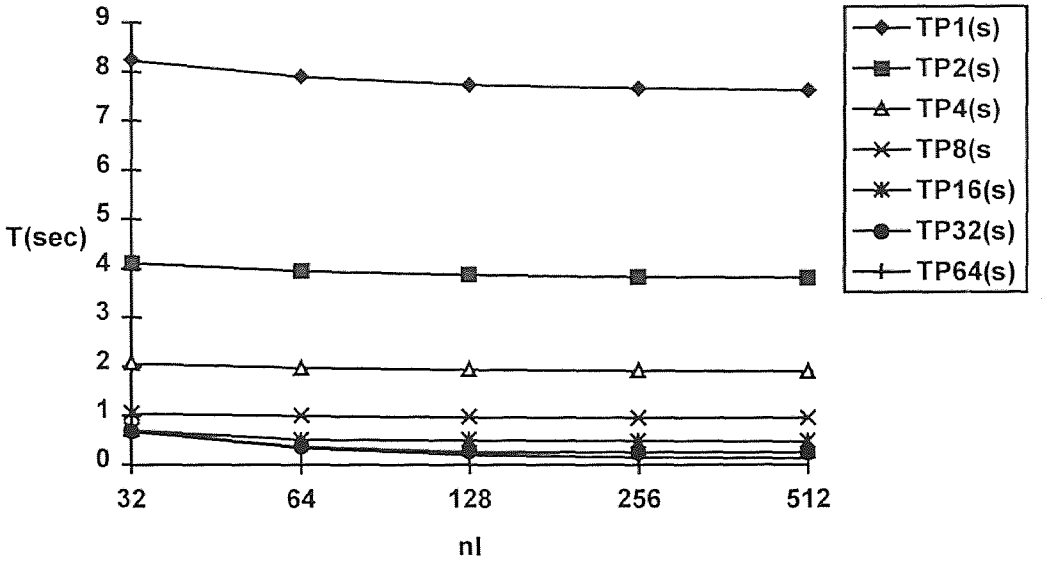

Figure B.7 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

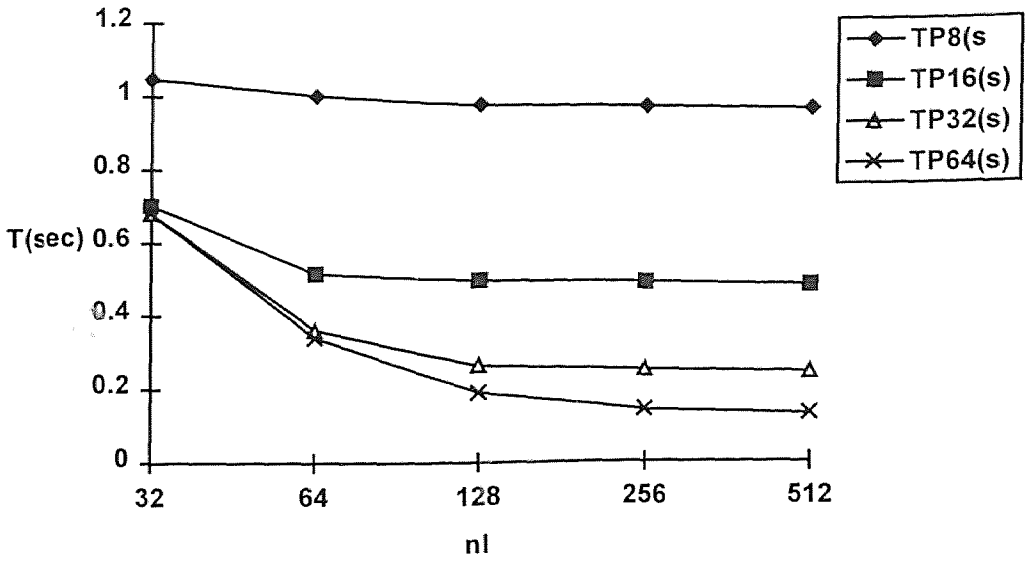

Figure B.8 Execution time vs. Shared-Bus-Width (*nl*) for P=8, 16, 32 and 64 when *template matrix*=6x6

Table B.6 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

Speedup vs. Shared-Bus-Width, *template matrix*=6x6							
nl	SP1	SP2	SP4	SP8(s	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.0431	1.0433	1.0445	1.0481	1.3711	1.8889	2.0000
128	1.0662	1.0666	1.0683	1.0795	1.4268	2.6054	3.5789
256	1.0781	1.0783	1.0806	1.0930	1.4564	2.7419	4.8571
512	1.0842	1.0843	1.0869	1.1011	1.4748	2.8099	5.2713

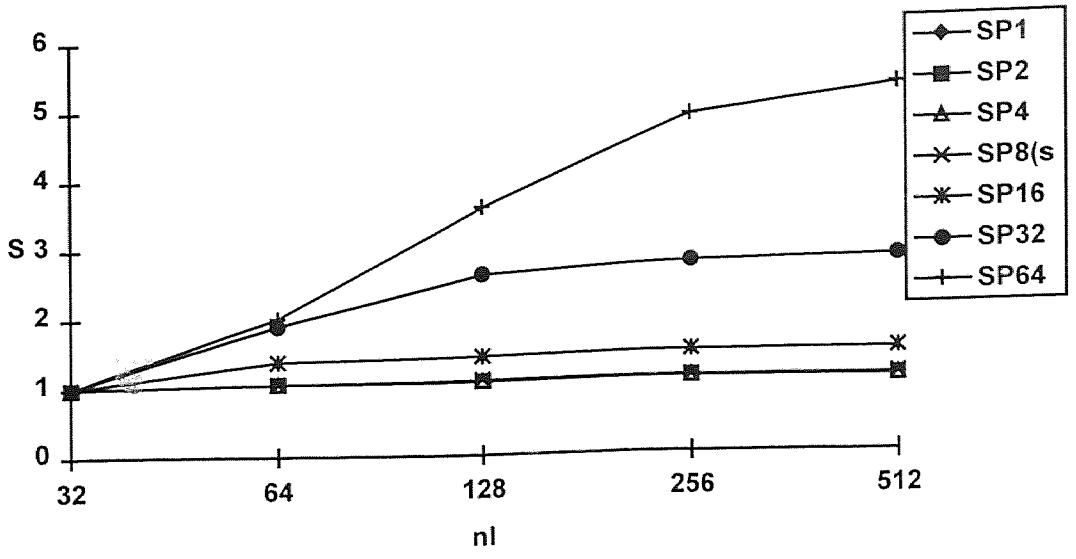

Figure B.9 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=6x6

Template matrix =9x9

Table B.7 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

Execution Time vs. Shared-Bus-Width, *template matrix*=9x9							
nl	TP1(s)	TP2(s)	TP4(s)	TP8(s	TP16(s)	TP32(s)	TP64(s)
32	17.701	8.851	4.432	2.230	1.146	0.767	0.710
64	17.344	8.672	4.339	2.177	1.104	0.584	0.410
128	17.166	8.583	4.293	2.150	1.083	0.557	0.311
256	17.077	8.538	4.270	2.137	1.072	0.544	0.288
512	17.033	8.516	4.258	2.130	1.067	0.537	0.277

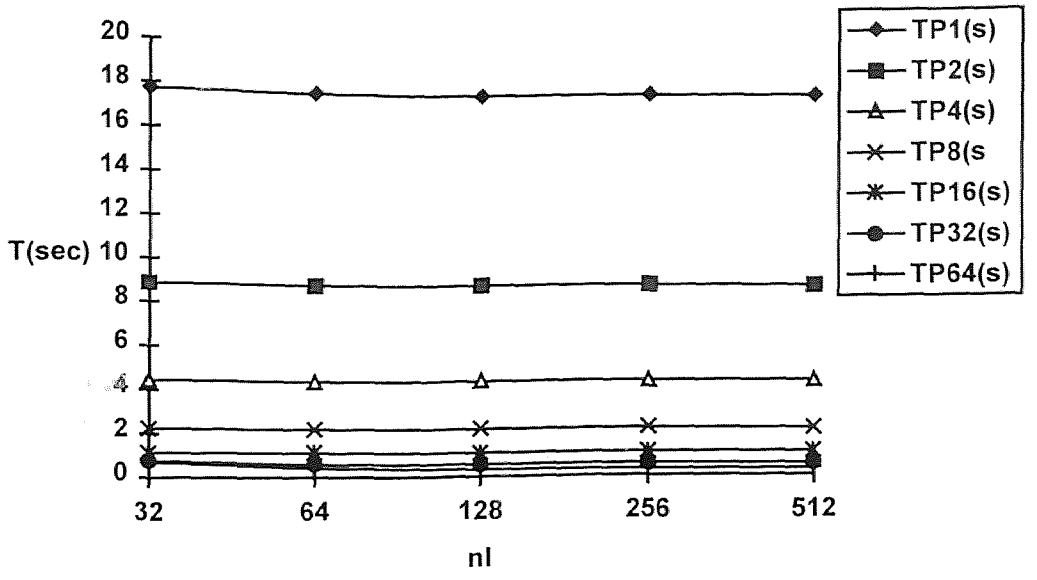

Figure B.10 Execution time vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

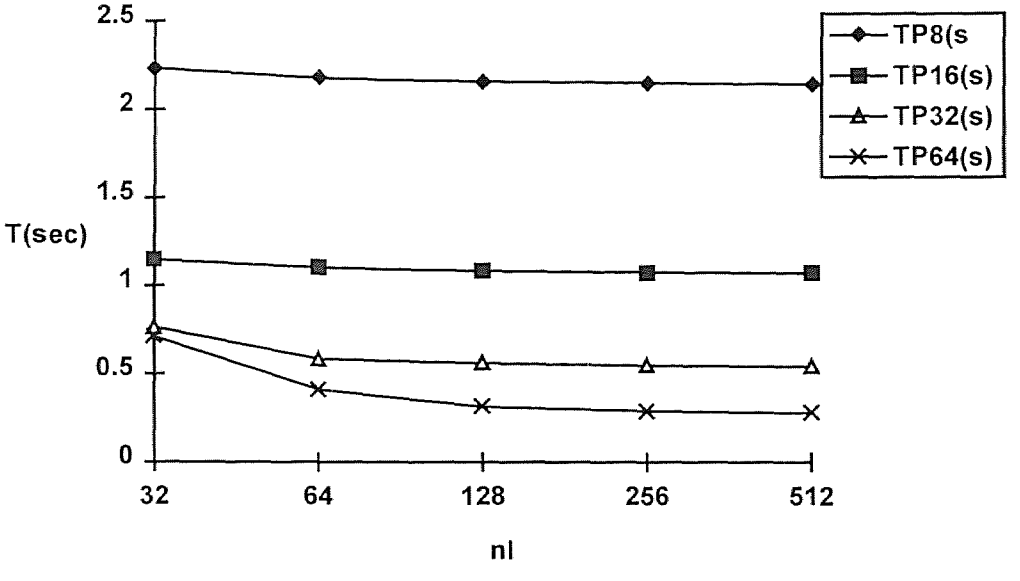

Figure B.11 Execution time vs. Shared-Bus-Width (*nl*) for P=8, 16, 32 and 64 when *template matrix*=9x9

Table B.8 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

Speedup vs. Shared-Bus-Width, *template matrix*=9x9							
nl	SP1	SP2	SP4	SP8(s	SP16	SP32	SP64
32	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
64	1.0206	1.0206	1.0214	1.0243	1.0380	1.3134	1.7317
128	1.0312	1.0312	1.0324	1.0372	1.0582	1.3770	2.2830
256	1.0365	1.0367	1.0379	1.0435	1.0690	1.4099	2.4653
512	1.0392	1.0393	1.0409	1.0469	1.0740	1.4283	2.5632

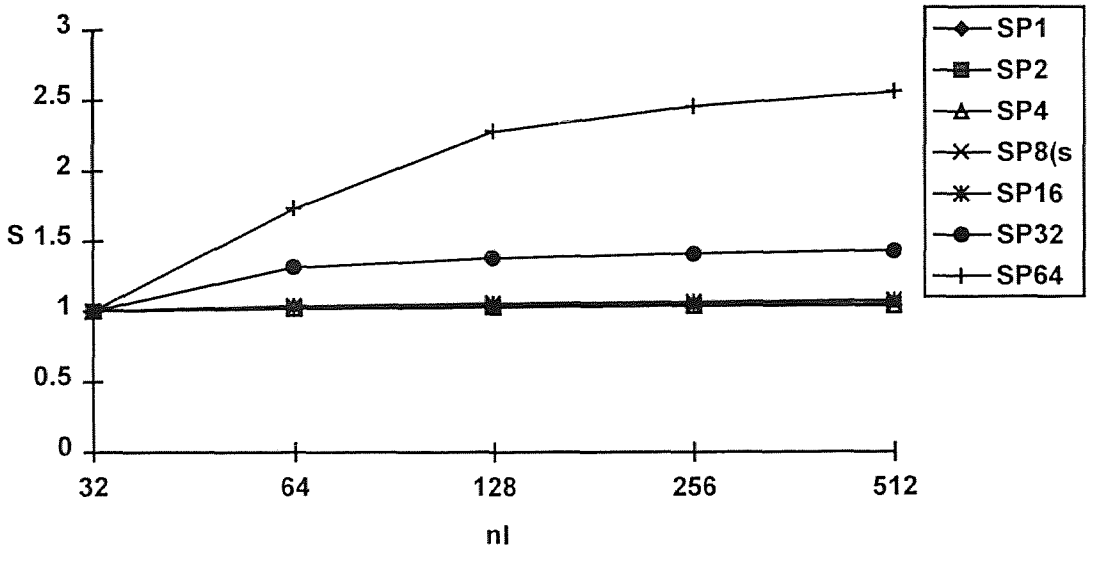

Figure B.12 Speedup vs. Shared-Bus-Width (*nl*) for P=1, 2, 4, 8, 16, 32 and 64 when *template matrix*=9x9

APPENDIX C

RESULTS

Appendix C provides results for all combinations of hardware and software tested. In order for one to be able to read the results one needs to know that *mr* stands for the number of rows of the *template matrix*, *nc* stands for the number of columns of the *template matrix*, P stands for the number of processors, *nl* stands for the *shared bus width*, *Twr* stands for *Twratio*, cacheusd stands for the size of the prefetching cache, T stands for the execution time of the application on the P-node system, *btlnck* stands for bottleneck, and *engh BUS BW* stands for enough shared bus bandwidth.

P=1

```
mr  nc  nl  numrowun  num_loads
2   2   32    1        8192.0 Twr < P ==>btlnck  Twr = 0.9    cacheusd(Kbytes)= 12.0    T3s(ms)= 1792.6913
mr  nc  nl  numrowun  num_loads
2   2   32    2        4096.0 Twr > P==>engh BUS BW Twr = 1.07   cacheusd(Kbytes)= 20.0    T3s(ms)= 1631.0401
mr  nc  nl  numrowun  num_loads
2   2   32    4        2048.0 Twr > P==>engh BUS BW Twr = 1.19   cacheusd(Kbytes)= 36.0    T3s(ms)= 1550.1186
mr  nc  nl  numrowun  num_loads
2   2   32    8        1024.0 Twr > P==>engh BUS BW Twr = 1.26   cacheusd(Kbytes)= 68.0    T3s(ms)= 1509.7731
mr  nc  nl  numrowun  num_loads
2   2   32    16        512.0 Twr > P==>engh BUS BW Twr = 1.29   cacheusd(Kbytes)= 132.1    T3s(ms)= 1489.8311
mr  nc  nl  numrowun  num_loads
2   2   32    32        256.0 Twr > P==>engh BUS BW Twr = 1.31   cacheusd(Kbytes)= 260.1    T3s(ms)= 1480.3212
mr  nc  nl  numrowun  num_loads
2   2   32    64        128.0 Twr > P==>engh BUS BW Twr = 1.32   cacheusd(Kbytes)= 516.3    T3s(ms)= 1476.4889
mr  nc  nl  numrowun  num_loads
2   2   32    128        64.0 Twr > P==>engh BUS BW Twr = 1.33   cacheusd(Kbytes)= 1028.5    T3s(ms)= 1476.4177
mr  nc  nl  numrowun  num_loads
2   2   32    256        32.0 Twr > P==>engh BUS BW Twr = 1.33   cacheusd(Kbytes)= 2053.0    T3s(ms)= 1480.0720
mr  nc  nl  numrowun  num_loads
2   2   32    512        16.0 Twr > P==>engh BUS BW Twr = 1.33   cacheusd(Kbytes)= 4102.0    T3s(ms)= 1489.2793
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   64    1        8192.0 Twr > P==>engh BUS BW Twr = 1.80   cacheusd(Kbytes)= 12.0    T3s(ms)= 1320.7937
mr  nc  nl  numrowun  num_loads
2   2   64    2        4096.0 Twr > P==>engh BUS BW Twr = 2.14   cacheusd(Kbytes)= 20.0    T3s(ms)= 1237.4593
mr  nc  nl  numrowun  num_loads
2   2   64    4        2048.0 Twr > P==>engh BUS BW Twr = 2.38   cacheusd(Kbytes)= 36.0    T3s(ms)= 1195.9745
mr  nc  nl  numrowun  num_loads
2   2   64    8        1024.0 Twr > P==>engh BUS BW Twr = 2.51   cacheusd(Kbytes)= 68.0    T3s(ms)= 1175.2130
mr  nc  nl  numrowun  num_loads
2   2   64    16        512.0 Twr > P==>engh BUS BW Twr = 2.59   cacheusd(Kbytes)= 132.1    T3s(ms)= 1164.9476
mr  nc  nl  numrowun  num_loads
2   2   64    32        256.0 Twr > P==>engh BUS BW Twr = 2.63   cacheusd(Kbytes)= 260.1    T3s(ms)= 1160.0455
mr  nc  nl  numrowun  num_loads
2   2   64    64        128.0 Twr > P==>engh BUS BW Twr = 2.65   cacheusd(Kbytes)= 516.3    T3s(ms)= 1158.0557
mr  nc  nl  numrowun  num_loads
2   2   64    128        64.0 Twr > P==>engh BUS BW Twr = 2.66   cacheusd(Kbytes)= 1028.5    T3s(ms)= 1157.9832
mr  nc  nl  numrowun  num_loads
2   2   64    256        32.0 Twr > P==>engh BUS BW Twr = 2.66   cacheusd(Kbytes)= 2053.0    T3s(ms)= 1159.7921
mr  nc  nl  numrowun  num_loads
2   2   64    512        16.0 Twr > P==>engh BUS BW Twr = 2.66   cacheusd(Kbytes)= 4102.0    T3s(ms)= 1164.3864
*********************************************************************************
```

```
 mr  nc  nl  numrowun  num_loads
2    2   128     1          8192.0  Twr > P==>engh BUS BW Twr = 3.59   cacheusd(Kbytes)= 12.0     T3s(ms)= 1084.8448
mr  nc  nl  numrowun  num_loads
2    2   128     2          4096.0  Twr > P==>engh BUS BW Twr = 4.29   cacheusd(Kbytes)= 20.0     T3s(ms)= 1040.8225
mr  nc  nl  numrowun  num_loads
2    2   128     4          2048.0  Twr > P==>engh BUS BW Twr = 4.75   cacheusd(Kbytes)= 36.0     T3s(ms)= 1018.8256
mr  nc  nl  numrowun  num_loads
2    2   128     8          1024.0  Twr > P==>engh BUS BW Twr = 5.03   cacheusd(Kbytes)= 68.0     T3s(ms)= 1007.9329
mr  nc  nl  numrowun  num_loads
2    2   128    16           512.0  Twr > P==>engh BUS BW Twr = 5.18   cacheusd(Kbytes)= 132.1    T3s(ms)= 1002.5058
mr  nc  nl  numrowun  num_loads
2    2   128    32           256.0  Twr > P==>engh BUS BW Twr = 5.25   cacheusd(Kbytes)= 260.1    T3s(ms)= 999.9076
mr  nc  nl  numrowun  num_loads
2    2   128    64           128.0  Twr > P==>engh BUS BW Twr = 5.29   cacheusd(Kbytes)= 516.3    T3s(ms)= 998.8391
mr  nc  nl  numrowun  num_loads
2    2   128   128            64.0  Twr > P==>engh BUS BW Twr = 5.31   cacheusd(Kbytes)= 1028.5   T3s(ms)= 998.7661
mr  nc  nl  numrowun  num_loads
2    2   128   256            32.0  Twr > P==>engh BUS BW Twr = 5.32   cacheusd(Kbytes)= 2053.0   T3s(ms)= 999.6521
mr  nc  nl  numrowun  num_loads
2    2   128   512            16.0  Twr > P==>engh BUS BW Twr = 5.33   cacheusd(Kbytes)= 4102.0   T3s(ms)= 1001.9401
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
2    2   256     1          8192.0  Twr > P==>engh BUS BW Twr = 7.15   cacheusd(Kbytes)= 12.0     T3s(ms)= 966.8705
mr  nc  nl  numrowun  num_loads
2    2   256     2          4096.0  Twr > P==>engh BUS BW Twr = 8.56   cacheusd(Kbytes)= 20.0     T3s(ms)= 942.5041
mr  nc  nl  numrowun  num_loads
2    2   256     4          2048.0  Twr > P==>engh BUS BW Twr = 9.50   cacheusd(Kbytes)= 36.0     T3s(ms)= 930.3281
mr  nc  nl  numrowun  num_loads
2    2   256     8          1024.0  Twr > P==>engh BUS BW Twr = 10.05  cacheusd(Kbytes)= 68.0     T3s(ms)= 924.2545
mr  nc  nl  numrowun  num_loads
2    2   256    16           512.0  Twr > P==>engh BUS BW Twr = 10.35  cacheusd(Kbytes)= 132.1    T3s(ms)= 921.2849
mr  nc  nl  numrowun  num_loads
2    2   256    32           256.0  Twr > P==>engh BUS BW Twr = 10.51  cacheusd(Kbytes)= 260.1    T3s(ms)= 919.8386
mr  nc  nl  numrowun  num_loads
2    2   256    64           128.0  Twr > P==>engh BUS BW Twr = 10.59  cacheusd(Kbytes)= 516.3    T3s(ms)= 919.2308
mr  nc  nl  numrowun  num_loads
2    2   256   128            64.0  Twr > P==>engh BUS BW Twr = 10.63  cacheusd(Kbytes)= 1028.5   T3s(ms)= 919.1575
mr  nc  nl  numrowun  num_loads
2    2   256   256            32.0  Twr > P==>engh BUS BW Twr = 10.65  cacheusd(Kbytes)= 2053.0   T3s(ms)= 919.5821
mr  nc  nl  numrowun  num_loads
2    2   256   512            16.0  Twr > P==>engh BUS BW Twr = 10.66  cacheusd(Kbytes)= 4102.0   T3s(ms)= 920.7169
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
2    2   512     1          8192.0  Twr > P==>engh BUS BW Twr = 14.23  cacheusd(Kbytes)= 12.0     T3s(ms)= 907.8833
mr  nc  nl  numrowun  num_loads
2    2   512     2          4096.0  Twr > P==>engh BUS BW Twr = 17.06  cacheusd(Kbytes)= 20.0     T3s(ms)= 893.3449
mr  nc  nl  numrowun  num_loads
2    2   512     4          2048.0  Twr > P==>engh BUS BW Twr = 18.96  cacheusd(Kbytes)= 36.0     T3s(ms)= 886.0793
mr  nc  nl  numrowun  num_loads
2    2   512     8          1024.0  Twr > P==>engh BUS BW Twr = 20.08  cacheusd(Kbytes)= 68.0     T3s(ms)= 882.4537
mr  nc  nl  numrowun  num_loads
2    2   512    16           512.0  Twr > P==>engh BUS BW Twr = 20.70  cacheusd(Kbytes)= 132.1    T3s(ms)= 880.6553
mr  nc  nl  numrowun  num_loads
2    2   512    32           256.0  Twr > P==>engh BUS BW Twr = 21.01  cacheusd(Kbytes)= 260.1    T3s(ms)= 879.8041
mr  nc  nl  numrowun  num_loads
2    2   512    64           128.0  Twr > P==>engh BUS BW Twr = 21.17  cacheusd(Kbytes)= 516.3    T3s(ms)= 879.4266
mr  nc  nl  numrowun  num_loads
2    2   512   128            64.0  Twr > P==>engh BUS BW Twr = 21.25  cacheusd(Kbytes)= 1028.5   T3s(ms)= 879.3532
mr  nc  nl  numrowun  num_loads
2    2   512   256            32.0  Twr > P==>engh BUS BW Twr = 21.30  cacheusd(Kbytes)= 2053.0   T3s(ms)= 879.5471
mr  nc  nl  numrowun  num_loads
2    2   512   512            16.0  Twr > P==>engh BUS BW Twr = 21.32  cacheusd(Kbytes)= 4102.0   T3s(ms)= 880.1053
*******************************************************************************
 *******************************************************************************
 mr  nc  nl  numrowun  num_loads
3    3    32     1          8192.0  Twr > P==>engh BUS BW Twr = 1.51   cacheusd(Kbytes)= 16.0     T3s(ms)= 3158.1314
mr  nc  nl  numrowun  num_loads
```

```
3   3   32   2          4096.0  Twr > P==>engh BUS BW Twr = 2.00   cacheusd(Kbytes)= 24.0     T3s(ms)= 2838.0675
mr  nc  nl  numrowun  num_loads
3   3   32   4          2048.0  Twr > P==>engh BUS BW Twr = 2.40   cacheusd(Kbytes)= 40.0     T3s(ms)= 2678.0932
mr  nc  nl  numrowun  num_loads
3   3   32   8          1024.0  Twr > P==>engh BUS BW Twr = 2.67   cacheusd(Kbytes)= 72.1     T3s(ms)= 2598.2215
mr  nc  nl  numrowun  num_loads
3   3   32   16          512.0  Twr > P==>engh BUS BW Twr = 2.82   cacheusd(Kbytes)= 136.1    T3s(ms)= 2558.5165
mr  nc  nl  numrowun  num_loads
3   3   32   32          256.0  Twr > P==>engh BUS BW Twr = 2.91   cacheusd(Kbytes)= 264.3    T3s(ms)= 2539.1257
mr  nc  nl  numrowun  num_loads
3   3   32   64          128.0  Twr > P==>engh BUS BW Twr = 2.95   cacheusd(Kbytes)= 520.5    T3s(ms)= 2530.3537
mr  nc  nl  numrowun  num_loads
3   3   32   128          64.0  Twr > P==>engh BUS BW Twr = 2.97   cacheusd(Kbytes)= 1033.0   T3s(ms)= 2527.8145
mr  nc  nl  numrowun  num_loads
3   3   32   256          32.0  Twr > P==>engh BUS BW Twr = 2.99   cacheusd(Kbytes)= 2058.0   T3s(ms)= 2530.2385
mr  nc  nl  numrowun  num_loads
3   3   32   512          16.0  Twr > P==>engh BUS BW Twr = 2.99   cacheusd(Kbytes)= 4108.0   T3s(ms)= 2538.8377
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   64   1          8192.0  Twr > P==>engh BUS BW Twr = 3.01   cacheusd(Kbytes)= 16.0     T3s(ms)= 2528.3137
mr  nc  nl  numrowun  num_loads
3   3   64   2          4096.0  Twr > P==>engh BUS BW Twr = 4.01   cacheusd(Kbytes)= 24.0     T3s(ms)= 2365.5169
mr  nc  nl  numrowun  num_loads
3   3   64   4          2048.0  Twr > P==>engh BUS BW Twr = 4.80   cacheusd(Kbytes)= 40.0     T3s(ms)= 2284.3010
mr  nc  nl  numrowun  num_loads
3   3   64   8          1024.0  Twr > P==>engh BUS BW Twr = 5.33   cacheusd(Kbytes)= 72.1     T3s(ms)= 2243.7508
mr  nc  nl  numrowun  num_loads
3   3   64   16          512.0  Twr > P==>engh BUS BW Twr = 5.64   cacheusd(Kbytes)= 136.1    T3s(ms)= 2223.5910
mr  nc  nl  numrowun  num_loads
3   3   64   32          256.0  Twr > P==>engh BUS BW Twr = 5.82   cacheusd(Kbytes)= 264.3    T3s(ms)= 2213.7421
mr  nc  nl  numrowun  num_loads
3   3   64   64          128.0  Twr > P==>engh BUS BW Twr = 5.90   cacheusd(Kbytes)= 520.5    T3s(ms)= 2209.2793
mr  nc  nl  numrowun  num_loads
3   3   64   128          64.0  Twr > P==>engh BUS BW Twr = 5.95   cacheusd(Kbytes)= 1033.0   T3s(ms)= 2207.9713
mr  nc  nl  numrowun  num_loads
3   3   64   256          32.0  Twr > P==>engh BUS BW Twr = 5.97   cacheusd(Kbytes)= 2058.0   T3s(ms)= 2209.1641
mr  nc  nl  numrowun  num_loads
3   3   64   512          16.0  Twr > P==>engh BUS BW Twr = 5.98   cacheusd(Kbytes)= 4108.0   T3s(ms)= 2213.4541
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   128  1          8192.0  Twr > P==>engh BUS BW Twr = 6.02   cacheusd(Kbytes)= 16.0     T3s(ms)= 2213.0977
mr  nc  nl  numrowun  num_loads
3   3   128  2          4096.0  Twr > P==>engh BUS BW Twr = 8.01   cacheusd(Kbytes)= 24.0     T3s(ms)= 2129.2416
mr  nc  nl  numrowun  num_loads
3   3   128  4          2048.0  Twr > P==>engh BUS BW Twr = 9.60   cacheusd(Kbytes)= 40.0     T3s(ms)= 2087.4817
mr  nc  nl  numrowun  num_loads
3   3   128  8          1024.0  Twr > P==>engh BUS BW Twr = 10.66  cacheusd(Kbytes)= 72.1     T3s(ms)= 2066.5538
mr  nc  nl  numrowun  num_loads
3   3   128  16          512.0  Twr > P==>engh BUS BW Twr = 11.29  cacheusd(Kbytes)= 136.1    T3s(ms)= 2056.1476
mr  nc  nl  numrowun  num_loads
3   3   128  32          256.0  Twr > P==>engh BUS BW Twr = 11.63  cacheusd(Kbytes)= 264.3    T3s(ms)= 2051.0599
mr  nc  nl  numrowun  num_loads
3   3   128  64          128.0  Twr > P==>engh BUS BW Twr = 11.81  cacheusd(Kbytes)= 520.5    T3s(ms)= 2048.7468
mr  nc  nl  numrowun  num_loads
3   3   128  128          64.0  Twr > P==>engh BUS BW Twr = 11.90  cacheusd(Kbytes)= 1033.0   T3s(ms)= 2048.0521
mr  nc  nl  numrowun  num_loads
3   3   128  256          32.0  Twr > P==>engh BUS BW Twr = 11.95  cacheusd(Kbytes)= 2058.0   T3s(ms)= 2048.6281
mr  nc  nl  numrowun  num_loads
3   3   128  512          16.0  Twr > P==>engh BUS BW Twr = 11.97  cacheusd(Kbytes)= 4108.0   T3s(ms)= 2050.7629
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   256  1          8192.0  Twr > P==>engh BUS BW Twr = 12.02  cacheusd(Kbytes)= 16.0     T3s(ms)= 2055.7969
mr  nc  nl  numrowun  num_loads
3   3   256  2          4096.0  Twr > P==>engh BUS BW Twr = 16.01  cacheusd(Kbytes)= 24.0     T3s(ms)= 2011.2577
mr  nc  nl  numrowun  num_loads
3   3   256  4          2048.0  Twr > P==>engh BUS BW Twr = 19.20  cacheusd(Kbytes)= 40.0     T3s(ms)= 1988.9953
mr  nc  nl  numrowun  num_loads
```

```
3   3   256   8          1024.0  Twr > P==>engh BUS BW Twr = 21.32   cacheusd(Kbytes)= 72.1    T3s(ms)= 1977.9553
mr  nc  nl  numrowun  num_loads
3   3   256   16          512.0  Twr > P==>engh BUS BW Twr = 22.57   cacheusd(Kbytes)= 136.1    T3s(ms)= 1972.4258
mr  nc  nl  numrowun  num_loads
3   3   256   32          256.0  Twr > P==>engh BUS BW Twr = 23.26   cacheusd(Kbytes)= 264.3    T3s(ms)= 1969.7188
mr  nc  nl  numrowun  num_loads
3   3   256   64          128.0  Twr > P==>engh BUS BW Twr = 23.61   cacheusd(Kbytes)= 520.5    T3s(ms)= 1968.4807
mr  nc  nl  numrowun  num_loads
3   3   256   128          64.0  Twr > P==>engh BUS BW Twr = 23.80   cacheusd(Kbytes)= 1033.0    T3s(ms)= 1968.0925
mr  nc  nl  numrowun  num_loads
3   3   256   256          32.0  Twr > P==>engh BUS BW Twr = 23.89   cacheusd(Kbytes)= 2058.0    T3s(ms)= 1968.3601
mr  nc  nl  numrowun  num_loads
3   3   256   512          16.0  Twr > P==>engh BUS BW Twr = 23.94   cacheusd(Kbytes)= 4108.0    T3s(ms)= 1969.4173
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   512   1          8192.0  Twr > P==>engh BUS BW Twr = 23.95   cacheusd(Kbytes)= 16.0    T3s(ms)= 1977.1465
mr  nc  nl  numrowun  num_loads
3   3   512   2          4096.0  Twr > P==>engh BUS BW Twr = 31.93   cacheusd(Kbytes)= 24.0    T3s(ms)= 1952.2657
mr  nc  nl  numrowun  num_loads
3   3   512   4          2048.0  Twr > P==>engh BUS BW Twr = 38.34   cacheusd(Kbytes)= 40.0    T3s(ms)= 1939.8289
mr  nc  nl  numrowun  num_loads
3   3   512   8          1024.0  Twr > P==>engh BUS BW Twr = 42.63   cacheusd(Kbytes)= 72.1    T3s(ms)= 1933.6177
mr  nc  nl  numrowun  num_loads
3   3   512   16          512.0  Twr > P==>engh BUS BW Twr = 45.12   cacheusd(Kbytes)= 136.1    T3s(ms)= 1930.5649
mr  nc  nl  numrowun  num_loads
3   3   512   32          256.0  Twr > P==>engh BUS BW Twr = 46.50   cacheusd(Kbytes)= 264.3    T3s(ms)= 1929.0482
mr  nc  nl  numrowun  num_loads
3   3   512   64          128.0  Twr > P==>engh BUS BW Twr = 47.22   cacheusd(Kbytes)= 520.5    T3s(ms)= 1928.3476
mr  nc  nl  numrowun  num_loads
3   3   512   128          64.0  Twr > P==>engh BUS BW Twr = 47.59   cacheusd(Kbytes)= 1033.0    T3s(ms)= 1928.1127
mr  nc  nl  numrowun  num_loads
3   3   512   256          32.0  Twr > P==>engh BUS BW Twr = 47.78   cacheusd(Kbytes)= 2058.0    T3s(ms)= 1928.2260
mr  nc  nl  numrowun  num_loads
3   3   512   512          16.0  Twr > P==>engh BUS BW Twr = 47.87   cacheusd(Kbytes)= 4108.0    T3s(ms)= 1928.7445
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   32   1          8192.0  Twr > P==>engh BUS BW Twr = 3.42   cacheusd(Kbytes)= 28.1    T3s(ms)= 9777.1787
mr  nc  nl  numrowun  num_loads
6   6   32   2          4096.0  Twr > P==>engh BUS BW Twr = 5.32   cacheusd(Kbytes)= 36.1    T3s(ms)= 8979.4189
mr  nc  nl  numrowun  num_loads
6   6   32   4          2048.0  Twr > P==>engh BUS BW Twr = 7.36   cacheusd(Kbytes)= 52.2    T3s(ms)= 8580.4433
mr  nc  nl  numrowun  num_loads
6   6   32   8          1024.0  Twr > P==>engh BUS BW Twr = 9.12   cacheusd(Kbytes)= 84.3    T3s(ms)= 8381.0713
mr  nc  nl  numrowun  num_loads
6   6   32   16          512.0  Twr > P==>engh BUS BW Twr = 10.35   cacheusd(Kbytes)= 148.4    T3s(ms)= 8281.6168
mr  nc  nl  numrowun  num_loads
6   6   32   32          256.0  Twr > P==>engh BUS BW Twr = 11.10   cacheusd(Kbytes)= 276.7    T3s(ms)= 8232.3525
mr  nc  nl  numrowun  num_loads
6   6   32   64          128.0  Twr > P==>engh BUS BW Twr = 11.52   cacheusd(Kbytes)= 533.3    T3s(ms)= 8208.6466
mr  nc  nl  numrowun  num_loads
6   6   32   128          64.0  Twr > P==>engh BUS BW Twr = 11.74   cacheusd(Kbytes)= 1046.6    T3s(ms)= 8198.6457
mr  nc  nl  numrowun  num_loads
6   6   32   256          32.0  Twr > P==>engh BUS BW Twr = 11.86   cacheusd(Kbytes)= 2073.1    T3s(ms)= 8197.3498
mr  nc  nl  numrowun  num_loads
6   6   32   512          16.0  Twr > P==>engh BUS BW Twr = 11.91   cacheusd(Kbytes)= 4126.1    T3s(ms)= 8204.1106
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   64   1          8192.0  Twr > P==>engh BUS BW Twr = 6.84   cacheusd(Kbytes)= 28.1    T3s(ms)= 8671.7574
mr  nc  nl  numrowun  num_loads
6   6   64   2          4096.0  Twr > P==>engh BUS BW Twr = 10.64   cacheusd(Kbytes)= 36.1    T3s(ms)= 8268.7303
mr  nc  nl  numrowun  num_loads
6   6   64   4          2048.0  Twr > P==>engh BUS BW Twr = 14.72   cacheusd(Kbytes)= 52.2    T3s(ms)= 8067.3993
mr  nc  nl  numrowun  num_loads
6   6   64   8          1024.0  Twr > P==>engh BUS BW Twr = 18.23   cacheusd(Kbytes)= 84.3    T3s(ms)= 7966.7149
mr  nc  nl  numrowun  num_loads
6   6   64   16          512.0  Twr > P==>engh BUS BW Twr = 20.70   cacheusd(Kbytes)= 148.4    T3s(ms)= 7916.4884
```

```
mr  nc  nl  numrowun  num_loads
6   6   64   32          256.0  Twr > P==>engh BUS BW Twr = 22.21   cacheusd(Kbytes)= 276.7    T3s(ms)= 7891.6067
mr  nc  nl  numrowun  num_loads
6   6   64   64          128.0  Twr > P==>engh BUS BW Twr = 23.04   cacheusd(Kbytes)= 533.3    T3s(ms)= 7879.6289
mr  nc  nl  numrowun  num_loads
6   6   64   128          64.0  Twr > P==>engh BUS BW Twr = 23.48   cacheusd(Kbytes)= 1046.6    T3s(ms)= 7874.5661
mr  nc  nl  numrowun  num_loads
6   6   64   256          32.0  Twr > P==>engh BUS BW Twr = 23.71   cacheusd(Kbytes)= 2073.1    T3s(ms)= 7873.8869
mr  nc  nl  numrowun  num_loads
6   6   64   512          16.0  Twr > P==>engh BUS BW Twr = 23.83   cacheusd(Kbytes)= 4126.1    T3s(ms)= 7877.2517
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   128   1          8192.0  Twr > P==>engh BUS BW Twr = 13.68   cacheusd(Kbytes)= 28.1    T3s(ms)= 8118.7395
mr  nc  nl  numrowun  num_loads
6   6   128   2          4096.0  Twr > P==>engh BUS BW Twr = 21.26   cacheusd(Kbytes)= 36.1    T3s(ms)= 7913.5396
mr  nc  nl  numrowun  num_loads
6   6   128   4          2048.0  Twr > P==>engh BUS BW Twr = 29.45   cacheusd(Kbytes)= 52.2    T3s(ms)= 7810.8004
mr  nc  nl  numrowun  num_loads
6   6   128   8          1024.0  Twr > P==>engh BUS BW Twr = 36.46   cacheusd(Kbytes)= 84.3    T3s(ms)= 7759.5367
mr  nc  nl  numrowun  num_loads
6   6   128   16          512.0  Twr > P==>engh BUS BW Twr = 41.40   cacheusd(Kbytes)= 148.4    T3s(ms)= 7733.9242
mr  nc  nl  numrowun  num_loads
6   6   128   32          256.0  Twr > P==>engh BUS BW Twr = 44.41   cacheusd(Kbytes)= 276.7    T3s(ms)= 7721.2338
mr  nc  nl  numrowun  num_loads
6   6   128   64          128.0  Twr > P==>engh BUS BW Twr = 46.08   cacheusd(Kbytes)= 533.3    T3s(ms)= 7715.1201
mr  nc  nl  numrowun  num_loads
6   6   128   128          64.0  Twr > P==>engh BUS BW Twr = 46.97   cacheusd(Kbytes)= 1046.6    T3s(ms)= 7712.5263
mr  nc  nl  numrowun  num_loads
6   6   128   256          32.0  Twr > P==>engh BUS BW Twr = 47.42   cacheusd(Kbytes)= 2073.1    T3s(ms)= 7712.1555
mr  nc  nl  numrowun  num_loads
6   6   128   512          16.0  Twr > P==>engh BUS BW Twr = 47.65   cacheusd(Kbytes)= 4126.1    T3s(ms)= 7713.8223
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   256   1          8192.0  Twr > P==>engh BUS BW Twr = 27.36   cacheusd(Kbytes)= 28.1    T3s(ms)= 7842.2305
mr  nc  nl  numrowun  num_loads
6   6   256   2          4096.0  Twr > P==>engh BUS BW Twr = 42.49   cacheusd(Kbytes)= 36.1    T3s(ms)= 7735.9442
mr  nc  nl  numrowun  num_loads
6   6   256   4          2048.0  Twr > P==>engh BUS BW Twr = 58.90   cacheusd(Kbytes)= 52.2    T3s(ms)= 7682.5010
mr  nc  nl  numrowun  num_loads
6   6   256   8          1024.0  Twr > P==>engh BUS BW Twr = 72.90   cacheusd(Kbytes)= 84.3    T3s(ms)= 7655.9476
mr  nc  nl  numrowun  num_loads
6   6   256   16          512.0  Twr > P==>engh BUS BW Twr = 82.80   cacheusd(Kbytes)= 148.4    T3s(ms)= 7642.6229
mr  nc  nl  numrowun  num_loads
6   6   256   32          256.0  Twr > P==>engh BUS BW Twr = 88.82   cacheusd(Kbytes)= 276.7    T3s(ms)= 7636.0377
mr  nc  nl  numrowun  num_loads
6   6   256   64          128.0  Twr > P==>engh BUS BW Twr = 92.16   cacheusd(Kbytes)= 533.3    T3s(ms)= 7632.8608
mr  nc  nl  numrowun  num_loads
6   6   256   128          64.0  Twr > P==>engh BUS BW Twr = 93.93   cacheusd(Kbytes)= 1046.6    T3s(ms)= 7631.5039
mr  nc  nl  numrowun  num_loads
6   6   256   256          32.0  Twr > P==>engh BUS BW Twr = 94.85   cacheusd(Kbytes)= 2073.1    T3s(ms)= 7631.2885
mr  nc  nl  numrowun  num_loads
6   6   256   512          16.0  Twr > P==>engh BUS BW Twr = 95.31   cacheusd(Kbytes)= 4126.1    T3s(ms)= 7632.1069
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   512   1          8192.0  Twr > P==>engh BUS BW Twr = 54.73   cacheusd(Kbytes)= 28.1    T3s(ms)= 7703.9761
mr  nc  nl  numrowun  num_loads
6   6   512   2          4096.0  Twr > P==>engh BUS BW Twr = 84.84   cacheusd(Kbytes)= 36.1    T3s(ms)= 7647.1465
mr  nc  nl  numrowun  num_loads
6   6   512   4          2048.0  Twr > P==>engh BUS BW Twr = 117.66   cacheusd(Kbytes)= 52.2    T3s(ms)= 7618.4281
mr  nc  nl  numrowun  num_loads
6   6   512   8          1024.0  Twr > P==>engh BUS BW Twr = 145.68   cacheusd(Kbytes)= 84.3    T3s(ms)= 7604.1530
mr  nc  nl  numrowun  num_loads
6   6   512   16          512.0  Twr > P==>engh BUS BW Twr = 165.53   cacheusd(Kbytes)= 148.4    T3s(ms)= 7596.9915
mr  nc  nl  numrowun  num_loads
6   6   512   32          256.0  Twr > P==>engh BUS BW Twr = 177.63   cacheusd(Kbytes)= 276.7    T3s(ms)= 7593.4396
mr  nc  nl  numrowun  num_loads
6   6   512   64          128.0  Twr > P==>engh BUS BW Twr = 184.33   cacheusd(Kbytes)= 533.3    T3s(ms)= 7591.7312
```

```
mr  nc  nl  numrowun  num_loads
6   6   512  128          64.0  Twr > P==>engh BUS BW Twr = 187.87   cacheusd(Kbytes)= 1046.6    T3s(ms)= 7590.9928
mr  nc  nl  numrowun  num_loads
6   6   512  256          32.0  Twr > P==>engh BUS BW Twr = 189.69   cacheusd(Kbytes)= 2073.1    T3s(ms)= 7590.8551
mr  nc  nl  numrowun  num_loads
6   6   512  512          16.0  Twr > P==>engh BUS BW Twr = 190.61   cacheusd(Kbytes)= 4126.1    T3s(ms)= 7591.2493
*********************************************************************************
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   32    1          8192.0  Twr > P==>engh BUS BW Twr = 5.37   cacheusd(Kbytes)= 40.3    T3s(ms)= 20180.3163
mr  nc  nl  numrowun  num_loads
9   9   32    2          4096.0  Twr > P==>engh BUS BW Twr = 8.95   cacheusd(Kbytes)= 48.3    T3s(ms)= 18899.9454
mr  nc  nl  numrowun  num_loads
9   9   32    4          2048.0  Twr > P==>engh BUS BW Twr = 13.43   cacheusd(Kbytes)= 64.4    T3s(ms)= 18259.8180
mr  nc  nl  numrowun  num_loads
9   9   32    8          1024.0  Twr > P==>engh BUS BW Twr = 17.91   cacheusd(Kbytes)= 96.5    T3s(ms)= 17939.8704
mr  nc  nl  numrowun  num_loads
9   9   32    16          512.0  Twr > P==>engh BUS BW Twr = 21.50   cacheusd(Kbytes)= 160.8    T3s(ms)= 17780.1288
mr  nc  nl  numrowun  num_loads
9   9   32    32          256.0  Twr > P==>engh BUS BW Twr = 23.90   cacheusd(Kbytes)= 289.2    T3s(ms)= 17700.7224
mr  nc  nl  numrowun  num_loads
9   9   32    64          128.0  Twr > P==>engh BUS BW Twr = 25.31   cacheusd(Kbytes)= 546.2    T3s(ms)= 17661.9480
mr  nc  nl  numrowun  num_loads
9   9   32    128          64.0  Twr > P==>engh BUS BW Twr = 26.08   cacheusd(Kbytes)= 1060.2    T3s(ms)= 17644.4184
mr  nc  nl  numrowun  num_loads
9   9   32    256          32.0  Twr > P==>engh BUS BW Twr = 26.48   cacheusd(Kbytes)= 2088.2    T3s(ms)= 17639.3688
mr  nc  nl  numrowun  num_loads
9   9   32    512          16.0  Twr > P==>engh BUS BW Twr = 26.69   cacheusd(Kbytes)= 4144.2    T3s(ms)= 17644.2744
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   64    1          8192.0  Twr > P==>engh BUS BW Twr = 10.74   cacheusd(Kbytes)= 40.3    T3s(ms)= 18596.2189
mr  nc  nl  numrowun  num_loads
9   9   64    2          4096.0  Twr > P==>engh BUS BW Twr = 17.90   cacheusd(Kbytes)= 48.3    T3s(ms)= 17949.8895
mr  nc  nl  numrowun  num_loads
9   9   64    4          2048.0  Twr > P==>engh BUS BW Twr = 26.85   cacheusd(Kbytes)= 64.4    T3s(ms)= 17626.7538
mr  nc  nl  numrowun  num_loads
9   9   64    8          1024.0  Twr > P==>engh BUS BW Twr = 35.82   cacheusd(Kbytes)= 96.5    T3s(ms)= 17465.2440
mr  nc  nl  numrowun  num_loads
9   9   64    16          512.0  Twr > P==>engh BUS BW Twr = 43.00   cacheusd(Kbytes)= 160.8    T3s(ms)= 17384.6052
mr  nc  nl  numrowun  num_loads
9   9   64    32          256.0  Twr > P==>engh BUS BW Twr = 47.80   cacheusd(Kbytes)= 289.2    T3s(ms)= 17344.5180
mr  nc  nl  numrowun  num_loads
9   9   64    64          128.0  Twr > P==>engh BUS BW Twr = 50.62   cacheusd(Kbytes)= 546.2    T3s(ms)= 17324.9388
mr  nc  nl  numrowun  num_loads
9   9   64    128          64.0  Twr > P==>engh BUS BW Twr = 52.16   cacheusd(Kbytes)= 1060.2    T3s(ms)= 17316.0780
mr  nc  nl  numrowun  num_loads
9   9   64    256          32.0  Twr > P==>engh BUS BW Twr = 52.96   cacheusd(Kbytes)= 2088.2    T3s(ms)= 17313.5052
mr  nc  nl  numrowun  num_loads
9   9   64    512          16.0  Twr > P==>engh BUS BW Twr = 53.37   cacheusd(Kbytes)= 4144.2    T3s(ms)= 17315.9340
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   128    1          8192.0  Twr > P==>engh BUS BW Twr = 21.48   cacheusd(Kbytes)= 40.3    T3s(ms)= 17804.1702
mr  nc  nl  numrowun  num_loads
9   9   128    2          4096.0  Twr > P==>engh BUS BW Twr = 35.79   cacheusd(Kbytes)= 48.3    T3s(ms)= 17474.8615
mr  nc  nl  numrowun  num_loads
9   9   128    4          2048.0  Twr > P==>engh BUS BW Twr = 53.71   cacheusd(Kbytes)= 64.4    T3s(ms)= 17310.2217
mr  nc  nl  numrowun  num_loads
9   9   128    8          1024.0  Twr > P==>engh BUS BW Twr = 71.64   cacheusd(Kbytes)= 96.5    T3s(ms)= 17227.9308
mr  nc  nl  numrowun  num_loads
9   9   128    16          512.0  Twr > P==>engh BUS BW Twr = 86.01   cacheusd(Kbytes)= 160.8    T3s(ms)= 17186.8434
mr  nc  nl  numrowun  num_loads
9   9   128    32          256.0  Twr > P==>engh BUS BW Twr = 95.59   cacheusd(Kbytes)= 289.2    T3s(ms)= 17166.4158
mr  nc  nl  numrowun  num_loads
9   9   128    64          128.0  Twr > P==>engh BUS BW Twr = 101.23   cacheusd(Kbytes)= 546.2    T3s(ms)= 17156.4342
mr  nc  nl  numrowun  num_loads
9   9   128    128          64.0  Twr > P==>engh BUS BW Twr = 104.31   cacheusd(Kbytes)= 1060.2    T3s(ms)= 17151.9078
mr  nc  nl  numrowun  num_loads
```

```
9   9   128   256          32.0  Twr > P==>engh BUS BW Twr = 105.92   cacheusd(Kbytes)= 2088.2    T3s(ms)= 17150.5734
mr  nc  nl  numrowun  num_loads
9   9   128   512          16.0  Twr > P==>engh BUS BW Twr = 106.75   cacheusd(Kbytes)= 4144.2    T3s(ms)= 17151.7638
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256    1          8192.0  Twr > P==>engh BUS BW Twr = 42.93   cacheusd(Kbytes)= 40.3    T3s(ms)= 17408.4532
mr  nc  nl  numrowun  num_loads
9   9   256    2          4096.0  Twr > P==>engh BUS BW Twr = 71.59   cacheusd(Kbytes)= 48.3    T3s(ms)= 17237.3476
mr  nc  nl  numrowun  num_loads
9   9   256    4          2048.0  Twr > P==>engh BUS BW Twr = 107.41   cacheusd(Kbytes)= 64.4    T3s(ms)= 17151.9556
mr  nc  nl  numrowun  num_loads
9   9   256    8          1024.0  Twr > P==>engh BUS BW Twr = 143.28   cacheusd(Kbytes)= 96.5    T3s(ms)= 17109.2742
mr  nc  nl  numrowun  num_loads
9   9   256   16           512.0  Twr > P==>engh BUS BW Twr = 172.02   cacheusd(Kbytes)= 160.8    T3s(ms)= 17087.9625
mr  nc  nl  numrowun  num_loads
9   9   256   32           256.0  Twr > P==>engh BUS BW Twr = 191.19   cacheusd(Kbytes)= 289.2    T3s(ms)= 17077.3647
mr  nc  nl  numrowun  num_loads
9   9   256   64           128.0  Twr > P==>engh BUS BW Twr = 202.47   cacheusd(Kbytes)= 546.2    T3s(ms)= 17072.1819
mr  nc  nl  numrowun  num_loads
9   9   256   128           64.0  Twr > P==>engh BUS BW Twr = 208.62   cacheusd(Kbytes)= 1060.2    T3s(ms)= 17069.8227
mr  nc  nl  numrowun  num_loads
9   9   256   256           32.0  Twr > P==>engh BUS BW Twr = 211.85   cacheusd(Kbytes)= 2088.2    T3s(ms)= 17069.1075
mr  nc  nl  numrowun  num_loads
9   9   256   512           16.0  Twr > P==>engh BUS BW Twr = 213.49   cacheusd(Kbytes)= 4144.2    T3s(ms)= 17069.6787
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   512    1          8192.0  Twr > P==>engh BUS BW Twr = 85.72   cacheusd(Kbytes)= 40.3    T3s(ms)= 17210.5946
mr  nc  nl  numrowun  num_loads
9   9   512    2          4096.0  Twr > P==>engh BUS BW Twr = 142.99   cacheusd(Kbytes)= 48.3    T3s(ms)= 17118.7442
mr  nc  nl  numrowun  num_loads
9   9   512    4          2048.0  Twr > P==>engh BUS BW Twr = 214.83   cacheusd(Kbytes)= 64.4    T3s(ms)= 17072.8226
mr  nc  nl  numrowun  num_loads
9   9   512    8          1024.0  Twr > P==>engh BUS BW Twr = 286.57   cacheusd(Kbytes)= 96.5    T3s(ms)= 17049.9459
mr  nc  nl  numrowun  num_loads
9   9   512   16           512.0  Twr > P==>engh BUS BW Twr = 344.03   cacheusd(Kbytes)= 160.8    T3s(ms)= 17038.5220
mr  nc  nl  numrowun  num_loads
9   9   512   32           256.0  Twr > P==>engh BUS BW Twr = 382.37   cacheusd(Kbytes)= 289.2    T3s(ms)= 17032.8391
mr  nc  nl  numrowun  num_loads
9   9   512   64           128.0  Twr > P==>engh BUS BW Twr = 404.94   cacheusd(Kbytes)= 546.2    T3s(ms)= 17030.0557
mr  nc  nl  numrowun  num_loads
9   9   512   128           64.0  Twr > P==>engh BUS BW Twr = 417.25   cacheusd(Kbytes)= 1060.2    T3s(ms)= 17028.7801
mr  nc  nl  numrowun  num_loads
9   9   512   256           32.0  Twr > P==>engh BUS BW Twr = 423.69   cacheusd(Kbytes)= 2088.2    T3s(ms)= 17028.3745
mr  nc  nl  numrowun  num_loads
9   9   512   512           16.0  Twr > P==>engh BUS BW Twr = 426.99   cacheusd(Kbytes)= 4144.2    T3s(ms)= 17028.6361
*********************************************************************************
 *********************************************************************************
```

P=2

```
mr  nc  nl  numrowun  num_loads
2   2   32    1          8192.0  Twr < P ==>btlnck  Twr = 0.9    cacheusd(kbytes)= 12.0    T4s(ms)= 945.2821
mr  nc  nl  numrowun  num_loads
2   2   32    2          4096.0  Twr < P ==>btlnck  Twr = 1.1    cacheusd(kbytes)= 20.0    T4s(ms)= 815.8841
mr  nc  nl  numrowun  num_loads
2   2   32    4          2048.0  Twr < P ==>btlnck  Twr = 1.2    cacheusd(kbytes)= 36.0    T4s(ms)= 775.3850
mr  nc  nl  numrowun  num_loads
2   2   32    8          1024.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 68.0    T4s(ms)= 755.2891
mr  nc  nl  numrowun  num_loads
2   2   32   16           512.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 132.1    T4s(ms)= 745.5486
mr  nc  nl  numrowun  num_loads
2   2   32   32           256.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 260.1    T4s(ms)= 741.2932
mr  nc  nl  numrowun  num_loads
2   2   32   64           128.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 516.3    T4s(ms)= 740.3952
mr  nc  nl  numrowun  num_loads
2   2   32   128           64.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 1028.5    T4s(ms)= 742.4056
mr  nc  nl  numrowun  num_loads
```

```
2   2   32   256          32.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 2053.0    T4s(ms)= 748.3296
mr  nc  nl  numrowun  num_loads
2   2   32   512          16.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 4102.0    T4s(ms)= 761.1292
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   64    1          8192.0  Twr < P ==>btlnck  Twr = 1.8    cacheusd(kbytes)= 12.0     T4s(ms)= 660.8184
mr  nc  nl  numrowun  num_loads
2   2   64    2          4096.0  Twr > P==>engh BUS BW Twr = 2.14   cacheusd(kbytes)= 20.0     T4s(ms)= 618.9438
mr  nc  nl  numrowun  num_loads
2   2   64    4          2048.0  Twr > P==>engh BUS BW Twr = 2.37   cacheusd(kbytes)= 36.0     T4s(ms)= 598.1182
mr  nc  nl  numrowun  num_loads
2   2   64    8          1024.0  Twr > P==>engh BUS BW Twr = 2.51   cacheusd(kbytes)= 68.0     T4s(ms)= 587.7246
mr  nc  nl  numrowun  num_loads
2   2   64    16          512.0  Twr > P==>engh BUS BW Twr = 2.59   cacheusd(kbytes)= 132.1    T4s(ms)= 582.6430
mr  nc  nl  numrowun  num_loads
2   2   64    32          256.0  Twr > P==>engh BUS BW Twr = 2.63   cacheusd(kbytes)= 260.1    T4s(ms)= 580.3326
mr  nc  nl  numrowun  num_loads
2   2   64    64          128.0  Twr > P==>engh BUS BW Twr = 2.65   cacheusd(kbytes)= 516.3    T4s(ms)= 579.6382
mr  nc  nl  numrowun  num_loads
2   2   64   128           64.0  Twr > P==>engh BUS BW Twr = 2.66   cacheusd(kbytes)= 1028.5    T4s(ms)= 580.2126
mr  nc  nl  numrowun  num_loads
2   2   64   256           32.0  Twr > P==>engh BUS BW Twr = 2.66   cacheusd(kbytes)= 2053.0    T4s(ms)= 582.3430
mr  nc  nl  numrowun  num_loads
2   2   64   512           16.0  Twr > P==>engh BUS BW Twr = 2.66   cacheusd(kbytes)= 4102.0    T4s(ms)= 587.0946
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   128   1          8192.0  Twr > P==>engh BUS BW Twr = 3.57   cacheusd(kbytes)= 12.0     T4s(ms)= 542.8318
mr  nc  nl  numrowun  num_loads
2   2   128   2          4096.0  Twr > P==>engh BUS BW Twr = 4.28   cacheusd(kbytes)= 20.0     T4s(ms)= 520.6206
mr  nc  nl  numrowun  num_loads
2   2   128   4          2048.0  Twr > P==>engh BUS BW Twr = 4.75   cacheusd(kbytes)= 36.0     T4s(ms)= 509.5294
mr  nc  nl  numrowun  num_loads
2   2   128   8          1024.0  Twr > P==>engh BUS BW Twr = 5.02   cacheusd(kbytes)= 68.0     T4s(ms)= 504.0510
mr  nc  nl  numrowun  num_loads
2   2   128   16          512.0  Twr > P==>engh BUS BW Twr = 5.17   cacheusd(kbytes)= 132.1    T4s(ms)= 501.3502
mr  nc  nl  numrowun  num_loads
2   2   128   32          256.0  Twr > P==>engh BUS BW Twr = 5.25   cacheusd(kbytes)= 260.1    T4s(ms)= 500.1150
mr  nc  nl  numrowun  num_loads
2   2   128   64          128.0  Twr > P==>engh BUS BW Twr = 5.29   cacheusd(kbytes)= 516.3    T4s(ms)= 499.7278
mr  nc  nl  numrowun  num_loads
2   2   128   128          64.0  Twr > P==>engh BUS BW Twr = 5.31   cacheusd(kbytes)= 1028.5    T4s(ms)= 499.9950
mr  nc  nl  numrowun  num_loads
2   2   128   256          32.0  Twr > P==>engh BUS BW Twr = 5.32   cacheusd(kbytes)= 2053.0    T4s(ms)= 501.0502
mr  nc  nl  numrowun  num_loads
2   2   128   512          16.0  Twr > P==>engh BUS BW Twr = 5.33   cacheusd(kbytes)= 4102.0    T4s(ms)= 503.4210
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   256   1          8192.0  Twr > P==>engh BUS BW Twr = 7.10   cacheusd(kbytes)= 12.0     T4s(ms)= 483.8446
mr  nc  nl  numrowun  num_loads
2   2   256   2          4096.0  Twr > P==>engh BUS BW Twr = 8.52   cacheusd(kbytes)= 20.0     T4s(ms)= 471.4590
mr  nc  nl  numrowun  num_loads
2   2   256   4          2048.0  Twr > P==>engh BUS BW Twr = 9.47   cacheusd(kbytes)= 36.0     T4s(ms)= 465.2734
mr  nc  nl  numrowun  num_loads
2   2   256   8          1024.0  Twr > P==>engh BUS BW Twr = 10.04   cacheusd(kbytes)= 68.0     T4s(ms)= 462.1950
mr  nc  nl  numrowun  num_loads
2   2   256   16          512.0  Twr > P==>engh BUS BW Twr = 10.34   cacheusd(kbytes)= 132.1    T4s(ms)= 460.7038
mr  nc  nl  numrowun  num_loads
2   2   256   32          256.0  Twr > P==>engh BUS BW Twr = 10.50   cacheusd(kbytes)= 260.1    T4s(ms)= 460.0062
mr  nc  nl  numrowun  num_loads
2   2   256   64          128.0  Twr > P==>engh BUS BW Twr = 10.59   cacheusd(kbytes)= 516.3    T4s(ms)= 459.7726
mr  nc  nl  numrowun  num_loads
2   2   256   128          64.0  Twr > P==>engh BUS BW Twr = 10.63   cacheusd(kbytes)= 1028.5    T4s(ms)= 459.8862
mr  nc  nl  numrowun  num_loads
2   2   256   256          32.0  Twr > P==>engh BUS BW Twr = 10.65   cacheusd(kbytes)= 2053.0    T4s(ms)= 460.4038
mr  nc  nl  numrowun  num_loads
2   2   256   512          16.0  Twr > P==>engh BUS BW Twr = 10.66   cacheusd(kbytes)= 4102.0    T4s(ms)= 461.5842
*****************************************************************************
```

```
 mr  nc  nl  numrowun  num_loads
2    2   512    1        8192.0  Twr > P==>engh BUS BW Twr = 14.04   cacheusd(kbytes)= 12.0    T4s(ms)= 454.3510
mr  nc  nl  numrowun  num_loads
2    2   512    2        4096.0  Twr > P==>engh BUS BW Twr = 16.92   cacheusd(kbytes)= 20.0    T4s(ms)= 446.8782
mr  nc  nl  numrowun  num_loads
2    2   512    4        2048.0  Twr > P==>engh BUS BW Twr = 18.87   cacheusd(kbytes)= 36.0    T4s(ms)= 443.1454
mr  nc  nl  numrowun  num_loads
2    2   512    8        1024.0  Twr > P==>engh BUS BW Twr = 20.03   cacheusd(kbytes)= 68.0    T4s(ms)= 441.2862
mr  nc  nl  numrowun  num_loads
2    2   512   16         512.0  Twr > P==>engh BUS BW Twr = 20.67   cacheusd(kbytes)= 132.1   T4s(ms)= 440.3710
mr  nc  nl  numrowun  num_loads
2    2   512   32         256.0  Twr > P==>engh BUS BW Twr = 20.99   cacheusd(kbytes)= 260.1   T4s(ms)= 439.9518
mr  nc  nl  numrowun  num_loads
2    2   512   64         128.0  Twr > P==>engh BUS BW Twr = 21.16   cacheusd(kbytes)= 516.3   T4s(ms)= 439.7950
mr  nc  nl  numrowun  num_loads
2    2   512  128          64.0  Twr > P==>engh BUS BW Twr = 21.25   cacheusd(kbytes)= 1028.5  T4s(ms)= 439.8318
mr  nc  nl  numrowun  num_loads
2    2   512  256          32.0  Twr > P==>engh BUS BW Twr = 21.29   cacheusd(kbytes)= 2053.0  T4s(ms)= 440.0806
mr  nc  nl  numrowun  num_loads
2    2   512  512          16.0  Twr > P==>engh BUS BW Twr = 21.32   cacheusd(kbytes)= 4102.0  T4s(ms)= 440.6658
*******************************************************************************
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
3    3   32    1        8192.0  Twr < P ==>btlnck  Twr = 1.5    cacheusd(kbytes)= 16.0    T4s(ms)= 1579.5707
mr  nc  nl  numrowun  num_loads
3    3   32    2        4096.0  Twr > P==>engh BUS BW Twr = 2.00   cacheusd(kbytes)= 24.0    T4s(ms)= 1419.2382
mr  nc  nl  numrowun  num_loads
3    3   32    4        2048.0  Twr > P==>engh BUS BW Twr = 2.40   cacheusd(kbytes)= 40.0    T4s(ms)= 1339.1870
mr  nc  nl  numrowun  num_loads
3    3   32    8        1024.0  Twr > P==>engh BUS BW Twr = 2.67   cacheusd(kbytes)= 72.1    T4s(ms)= 1299.2766
mr  nc  nl  numrowun  num_loads
3    3   32   16         512.0  Twr > P==>engh BUS BW Twr = 2.82   cacheusd(kbytes)= 136.1   T4s(ms)= 1279.5518
mr  nc  nl  numrowun  num_loads
3    3   32   32         256.0  Twr > P==>engh BUS BW Twr = 2.91   cacheusd(kbytes)= 264.3   T4s(ms)= 1270.1502
mr  nc  nl  numrowun  num_loads
3    3   32   64         128.0  Twr > P==>engh BUS BW Twr = 2.95   cacheusd(kbytes)= 520.5   T4s(ms)= 1266.3710
mr  nc  nl  numrowun  num_loads
3    3   32  128          64.0  Twr > P==>engh BUS BW Twr = 2.97   cacheusd(kbytes)= 1033.0   T4s(ms)= 1266.3246
mr  nc  nl  numrowun  num_loads
3    3   32  256          32.0  Twr > P==>engh BUS BW Twr = 2.99   cacheusd(kbytes)= 2058.0   T4s(ms)= 1269.9878
mr  nc  nl  numrowun  num_loads
3    3   32  512          16.0  Twr > P==>engh BUS BW Twr = 2.99   cacheusd(kbytes)= 4108.0   T4s(ms)= 1279.1922
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
3    3   64    1        8192.0  Twr > P==>engh BUS BW Twr = 3.01   cacheusd(kbytes)= 16.0    T4s(ms)= 1264.5566
mr  nc  nl  numrowun  num_loads
3    3   64    2        4096.0  Twr > P==>engh BUS BW Twr = 4.00   cacheusd(kbytes)= 24.0    T4s(ms)= 1182.9630
mr  nc  nl  numrowun  num_loads
3    3   64    4        2048.0  Twr > P==>engh BUS BW Twr = 4.80   cacheusd(kbytes)= 40.0    T4s(ms)= 1142.2718
mr  nc  nl  numrowun  num_loads
3    3   64    8        1024.0  Twr > P==>engh BUS BW Twr = 5.33   cacheusd(kbytes)= 72.1    T4s(ms)= 1121.9838
mr  nc  nl  numrowun  num_loads
3    3   64   16         512.0  Twr > P==>engh BUS BW Twr = 5.64   cacheusd(kbytes)= 136.1   T4s(ms)= 1111.9550
mr  nc  nl  numrowun  num_loads
3    3   64   32         256.0  Twr > P==>engh BUS BW Twr = 5.81   cacheusd(kbytes)= 264.3   T4s(ms)= 1107.1710
mr  nc  nl  numrowun  num_loads
3    3   64   64         128.0  Twr > P==>engh BUS BW Twr = 5.90   cacheusd(kbytes)= 520.5   T4s(ms)= 1105.2398
mr  nc  nl  numrowun  num_loads
3    3   64  128          64.0  Twr > P==>engh BUS BW Twr = 5.95   cacheusd(kbytes)= 1033.0   T4s(ms)= 1105.1958
mr  nc  nl  numrowun  num_loads
3    3   64  256          32.0  Twr > P==>engh BUS BW Twr = 5.97   cacheusd(kbytes)= 2058.0   T4s(ms)= 1107.0170
mr  nc  nl  numrowun  num_loads
3    3   64  512          16.0  Twr > P==>engh BUS BW Twr = 5.98   cacheusd(kbytes)= 4108.0   T4s(ms)= 1111.6140
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
3    3  128    1        8192.0  Twr > P==>engh BUS BW Twr = 6.01   cacheusd(kbytes)= 16.0    T4s(ms)= 1106.9534
mr  nc  nl  numrowun  num_loads
```

```
3   3   128   2          4096.0  Twr > P==>engh BUS BW Twr = 8.00   cacheusd(kbytes)= 24.0    T4s(ms)= 1064.8254
mr  nc  nl  numrowun  num_loads
3   3   128   4          2048.0  Twr > P==>engh BUS BW Twr = 9.59   cacheusd(kbytes)= 40.0    T4s(ms)= 1043.8526
mr  nc  nl  numrowun  num_loads
3   3   128   8          1024.0  Twr > P==>engh BUS BW Twr = 10.66  cacheusd(kbytes)= 72.1    T4s(ms)= 1033.3566
mr  nc  nl  numrowun  num_loads
3   3   128   16          512.0  Twr > P==>engh BUS BW Twr = 11.29  cacheusd(kbytes)= 136.1   T4s(ms)= 1028.1662
mr  nc  nl  numrowun  num_loads
3   3   128   32          256.0  Twr > P==>engh BUS BW Twr = 11.63  cacheusd(kbytes)= 264.3   T4s(ms)= 1025.6862
mr  nc  nl  numrowun  num_loads
3   3   128   64          128.0  Twr > P==>engh BUS BW Twr = 11.81  cacheusd(kbytes)= 520.5   T4s(ms)= 1024.6766
mr  nc  nl  numrowun  num_loads
3   3   128   128          64.0  Twr > P==>engh BUS BW Twr = 11.90  cacheusd(kbytes)= 1033.0   T4s(ms)= 1024.6326
mr  nc  nl  numrowun  num_loads
3   3   128   256          32.0  Twr > P==>engh BUS BW Twr = 11.95  cacheusd(kbytes)= 2058.0   T4s(ms)= 1025.5322
mr  nc  nl  numrowun  num_loads
3   3   128   512          16.0  Twr > P==>engh BUS BW Twr = 11.97  cacheusd(kbytes)= 4108.0   T4s(ms)= 1027.8252
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   256   1          8192.0  Twr > P==>engh BUS BW Twr = 11.96  cacheusd(kbytes)= 16.0    T4s(ms)= 1028.3054
mr  nc  nl  numrowun  num_loads
3   3   256   2          4096.0  Twr > P==>engh BUS BW Twr = 15.95  cacheusd(kbytes)= 24.0    T4s(ms)= 1005.8334
mr  nc  nl  numrowun  num_loads
3   3   256   4          2048.0  Twr > P==>engh BUS BW Twr = 19.16  cacheusd(kbytes)= 40.0    T4s(ms)= 994.6046
mr  nc  nl  numrowun  num_loads
3   3   256   8          1024.0  Twr > P==>engh BUS BW Twr = 21.29  cacheusd(kbytes)= 72.1    T4s(ms)= 989.0430
mr  nc  nl  numrowun  num_loads
3   3   256   16          512.0  Twr > P==>engh BUS BW Twr = 22.56  cacheusd(kbytes)= 136.1   T4s(ms)= 986.2718
mr  nc  nl  numrowun  num_loads
3   3   256   32          256.0  Twr > P==>engh BUS BW Twr = 23.25  cacheusd(kbytes)= 264.3   T4s(ms)= 984.9438
mr  nc  nl  numrowun  num_loads
3   3   256   64          128.0  Twr > P==>engh BUS BW Twr = 23.61  cacheusd(kbytes)= 520.5   T4s(ms)= 984.3950
mr  nc  nl  numrowun  num_loads
3   3   256   128          64.0  Twr > P==>engh BUS BW Twr = 23.80  cacheusd(kbytes)= 1033.0   T4s(ms)= 984.3510
mr  nc  nl  numrowun  num_loads
3   3   256   256          32.0  Twr > P==>engh BUS BW Twr = 23.89  cacheusd(kbytes)= 2058.0   T4s(ms)= 984.7898
mr  nc  nl  numrowun  num_loads
3   3   256   512          16.0  Twr > P==>engh BUS BW Twr = 23.94  cacheusd(kbytes)= 4108.0   T4s(ms)= 985.9308
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   512   1          8192.0  Twr > P==>engh BUS BW Twr = 23.70  cacheusd(kbytes)= 16.0    T4s(ms)= 988.9814
mr  nc  nl  numrowun  num_loads
3   3   512   2          4096.0  Twr > P==>engh BUS BW Twr = 31.71  cacheusd(kbytes)= 24.0    T4s(ms)= 976.3374
mr  nc  nl  numrowun  num_loads
3   3   512   4          2048.0  Twr > P==>engh BUS BW Twr = 38.18  cacheusd(kbytes)= 40.0    T4s(ms)= 970.0190
mr  nc  nl  numrowun  num_loads
3   3   512   8          1024.0  Twr > P==>engh BUS BW Twr = 42.54  cacheusd(kbytes)= 72.1    T4s(ms)= 966.8670
mr  nc  nl  numrowun  num_loads
3   3   512   16          512.0  Twr > P==>engh BUS BW Twr = 45.07  cacheusd(kbytes)= 136.1   T4s(ms)= 965.3246
mr  nc  nl  numrowun  num_loads
3   3   512   32          256.0  Twr > P==>engh BUS BW Twr = 46.47  cacheusd(kbytes)= 264.3   T4s(ms)= 964.5726
mr  nc  nl  numrowun  num_loads
3   3   512   64          128.0  Twr > P==>engh BUS BW Twr = 47.21  cacheusd(kbytes)= 520.5   T4s(ms)= 964.2542
mr  nc  nl  numrowun  num_loads
3   3   512   128          64.0  Twr > P==>engh BUS BW Twr = 47.59  cacheusd(kbytes)= 1033.0   T4s(ms)= 964.2102
mr  nc  nl  numrowun  num_loads
3   3   512   256          32.0  Twr > P==>engh BUS BW Twr = 47.78  cacheusd(kbytes)= 2058.0   T4s(ms)= 964.4186
mr  nc  nl  numrowun  num_loads
3   3   512   512          16.0  Twr > P==>engh BUS BW Twr = 47.87  cacheusd(kbytes)= 4108.0   T4s(ms)= 964.9836
*********************************************************************************
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   32    1          8192.0  Twr > P==>engh BUS BW Twr = 3.42   cacheusd(kbytes)= 28.1    T4s(ms)= 4888.9214
mr  nc  nl  numrowun  num_loads
6   6   32    2          4096.0  Twr > P==>engh BUS BW Twr = 5.32   cacheusd(kbytes)= 36.1    T4s(ms)= 4489.8558
mr  nc  nl  numrowun  num_loads
6   6   32    4          2048.0  Twr > P==>engh BUS BW Twr = 7.36   cacheusd(kbytes)= 52.2    T4s(ms)= 4290.3038
```

```
mr  nc  nl  numrowun  num_loads
6   6   32    8          1024.0  Twr > P==>engh BUS BW Twr = 9.12   cacheusd(kbytes)= 84.3     T4s(ms)= 4190.6430
mr  nc  nl  numrowun  num_loads
6   6   32   16           512.0  Twr > P==>engh BUS BW Twr = 10.35   cacheusd(kbytes)= 148.4    T4s(ms)= 4141.0430
mr  nc  nl  numrowun  num_loads
6   6   32   32           256.0  Twr > P==>engh BUS BW Twr = 11.10   cacheusd(kbytes)= 276.7    T4s(ms)= 4116.7038
mr  nc  nl  numrowun  num_loads
6   6   32   64           128.0  Twr > P==>engh BUS BW Twr = 11.52   cacheusd(kbytes)= 533.3    T4s(ms)= 4105.4558
mr  nc  nl  numrowun  num_loads
6   6   32   128           64.0  Twr > P==>engh BUS BW Twr = 11.74   cacheusd(kbytes)= 1046.6    T4s(ms)= 4101.6750
mr  nc  nl  numrowun  num_loads
6   6   32   256           32.0  Twr > P==>engh BUS BW Twr = 11.86   cacheusd(kbytes)= 2073.1    T4s(ms)= 4103.4710
mr  nc  nl  numrowun  num_loads
6   6   32   512           16.0  Twr > P==>engh BUS BW Twr = 11.91   cacheusd(kbytes)= 4126.1    T4s(ms)= 4111.7418
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6   64    1          8192.0  Twr > P==>engh BUS BW Twr = 6.84   cacheusd(kbytes)= 28.1     T4s(ms)= 4336.2494
mr  nc  nl  numrowun  num_loads
6   6   64    2          4096.0  Twr > P==>engh BUS BW Twr = 10.63   cacheusd(kbytes)= 36.1    T4s(ms)= 4134.5406
mr  nc  nl  numrowun  num_loads
6   6   64    4          2048.0  Twr > P==>engh BUS BW Twr = 14.72   cacheusd(kbytes)= 52.2    T4s(ms)= 4033.7918
mr  nc  nl  numrowun  num_loads
6   6   64    8          1024.0  Twr > P==>engh BUS BW Twr = 18.23   cacheusd(kbytes)= 84.3    T4s(ms)= 3983.4366
mr  nc  nl  numrowun  num_loads
6   6   64   16           512.0  Twr > P==>engh BUS BW Twr = 20.70   cacheusd(kbytes)= 148.4    T4s(ms)= 3958.3742
mr  nc  nl  numrowun  num_loads
6   6   64   32           256.0  Twr > P==>engh BUS BW Twr = 22.20   cacheusd(kbytes)= 276.7    T4s(ms)= 3946.0734
mr  nc  nl  numrowun  num_loads
6   6   64   64           128.0  Twr > P==>engh BUS BW Twr = 23.04   cacheusd(kbytes)= 533.3    T4s(ms)= 3940.3838
mr  nc  nl  numrowun  num_loads
6   6   64   128           64.0  Twr > P==>engh BUS BW Twr = 23.48   cacheusd(kbytes)= 1046.6    T4s(ms)= 3938.4606
mr  nc  nl  numrowun  num_loads
6   6   64   256           32.0  Twr > P==>engh BUS BW Twr = 23.71   cacheusd(kbytes)= 2073.1    T4s(ms)= 3939.3422
mr  nc  nl  numrowun  num_loads
6   6   64   512           16.0  Twr > P==>engh BUS BW Twr = 23.83   cacheusd(kbytes)= 4126.1    T4s(ms)= 3943.4694
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6  128    1          8192.0  Twr > P==>engh BUS BW Twr = 13.66   cacheusd(kbytes)= 28.1     T4s(ms)= 4059.7598
mr  nc  nl  numrowun  num_loads
6   6  128    2          4096.0  Twr > P==>engh BUS BW Twr = 21.24   cacheusd(kbytes)= 36.1     T4s(ms)= 3956.9598
mr  nc  nl  numrowun  num_loads
6   6  128    4          2048.0  Twr > P==>engh BUS BW Twr = 29.43   cacheusd(kbytes)= 52.2     T4s(ms)= 3905.4974
mr  nc  nl  numrowun  num_loads
6   6  128    8          1024.0  Twr > P==>engh BUS BW Twr = 36.44   cacheusd(kbytes)= 84.3     T4s(ms)= 3879.8334
mr  nc  nl  numrowun  num_loads
6   6  128   16           512.0  Twr > P==>engh BUS BW Twr = 41.39   cacheusd(kbytes)= 148.4    T4s(ms)= 3867.0398
mr  nc  nl  numrowun  num_loads
6   6  128   32           256.0  Twr > P==>engh BUS BW Twr = 44.40   cacheusd(kbytes)= 276.7    T4s(ms)= 3860.7582
mr  nc  nl  numrowun  num_loads
6   6  128   64           128.0  Twr > P==>engh BUS BW Twr = 46.08   cacheusd(kbytes)= 533.3    T4s(ms)= 3857.8478
mr  nc  nl  numrowun  num_loads
6   6  128   128           64.0  Twr > P==>engh BUS BW Twr = 46.97   cacheusd(kbytes)= 1046.6    T4s(ms)= 3856.8534
mr  nc  nl  numrowun  num_loads
6   6  128   256           32.0  Twr > P==>engh BUS BW Twr = 47.42   cacheusd(kbytes)= 2073.1    T4s(ms)= 3857.2778
mr  nc  nl  numrowun  num_loads
6   6  128   512           16.0  Twr > P==>engh BUS BW Twr = 47.65   cacheusd(kbytes)= 4126.1    T4s(ms)= 3859.3332
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6  256    1          8192.0  Twr > P==>engh BUS BW Twr = 27.28   cacheusd(kbytes)= 28.1     T4s(ms)= 3921.5150
mr  nc  nl  numrowun  num_loads
6   6  256    2          4096.0  Twr > P==>engh BUS BW Twr = 42.39   cacheusd(kbytes)= 36.1     T4s(ms)= 3868.1694
mr  nc  nl  numrowun  num_loads
6   6  256    4          2048.0  Twr > P==>engh BUS BW Twr = 58.81   cacheusd(kbytes)= 52.2     T4s(ms)= 3841.3502
mr  nc  nl  numrowun  num_loads
6   6  256    8          1024.0  Twr > P==>engh BUS BW Twr = 72.82   cacheusd(kbytes)= 84.3     T4s(ms)= 3828.0318
mr  nc  nl  numrowun  num_loads
6   6  256   16           512.0  Twr > P==>engh BUS BW Twr = 82.75   cacheusd(kbytes)= 148.4    T4s(ms)= 3821.3630
```

```
mr  nc  nl  numrowun  num_loads
6   6   256   32        256.0  Twr > P==>engh BUS BW Twr = 88.79   cacheusd(kbytes)= 276.7    T4s(ms)= 3818.0958
mr  nc  nl  numrowun  num_loads
6   6   256   64        128.0  Twr > P==>engh BUS BW Twr = 92.15   cacheusd(kbytes)= 533.3    T4s(ms)= 3816.5774
mr  nc  nl  numrowun  num_loads
6   6   256   128        64.0  Twr > P==>engh BUS BW Twr = 93.93   cacheusd(kbytes)= 1046.6   T4s(ms)= 3816.0486
mr  nc  nl  numrowun  num_loads
6   6   256   256        32.0  Twr > P==>engh BUS BW Twr = 94.84   cacheusd(kbytes)= 2073.1   T4s(ms)= 3816.2450
mr  nc  nl  numrowun  num_loads
6   6   256   512        16.0  Twr > P==>engh BUS BW Twr = 95.31   cacheusd(kbytes)= 4126.1   T4s(ms)= 3817.2648
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   512   1         8192.0  Twr > P==>engh BUS BW Twr = 54.41   cacheusd(kbytes)= 28.1    T4s(ms)= 3852.3926
mr  nc  nl  numrowun  num_loads
6   6   512   2         4096.0  Twr > P==>engh BUS BW Twr = 84.45   cacheusd(kbytes)= 36.1    T4s(ms)= 3823.7742
mr  nc  nl  numrowun  num_loads
6   6   512   4         2048.0  Twr > P==>engh BUS BW Twr = 117.28  cacheusd(kbytes)= 52.2    T4s(ms)= 3809.3150
mr  nc  nl  numrowun  num_loads
6   6   512   8         1024.0  Twr > P==>engh BUS BW Twr = 145.40  cacheusd(kbytes)= 84.3    T4s(ms)= 3802.1310
mr  nc  nl  numrowun  num_loads
6   6   512   16         512.0  Twr > P==>engh BUS BW Twr = 165.34  cacheusd(kbytes)= 148.4   T4s(ms)= 3798.5342
mr  nc  nl  numrowun  num_loads
6   6   512   32         256.0  Twr > P==>engh BUS BW Twr = 177.52  cacheusd(kbytes)= 276.7   T4s(ms)= 3796.7646
mr  nc  nl  numrowun  num_loads
6   6   512   64         128.0  Twr > P==>engh BUS BW Twr = 184.27  cacheusd(kbytes)= 533.3   T4s(ms)= 3795.9422
mr  nc  nl  numrowun  num_loads
6   6   512   128         64.0  Twr > P==>engh BUS BW Twr = 187.84  cacheusd(kbytes)= 1046.6  T4s(ms)= 3795.6462
mr  nc  nl  numrowun  num_loads
6   6   512   256         32.0  Twr > P==>engh BUS BW Twr = 189.67  cacheusd(kbytes)= 2073.1  T4s(ms)= 3795.7286
mr  nc  nl  numrowun  num_loads
6   6   512   512         16.0  Twr > P==>engh BUS BW Twr = 190.61  cacheusd(kbytes)= 4126.1  T4s(ms)= 3796.2306
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   32    1         8192.0  Twr > P==>engh BUS BW Twr = 5.37   cacheusd(kbytes)= 40.3    T4s(ms)= 10090.4318
mr  nc  nl  numrowun  num_loads
9   9   32    2         4096.0  Twr > P==>engh BUS BW Twr = 8.95   cacheusd(kbytes)= 48.3    T4s(ms)= 9450.0606
mr  nc  nl  numrowun  num_loads
9   9   32    4         2048.0  Twr > P==>engh BUS BW Twr = 13.42  cacheusd(kbytes)= 64.4    T4s(ms)= 9129.9326
mr  nc  nl  numrowun  num_loads
9   9   32    8         1024.0  Twr > P==>engh BUS BW Twr = 17.91  cacheusd(kbytes)= 96.5    T4s(ms)= 8969.9838
mr  nc  nl  numrowun  num_loads
9   9   32    16         512.0  Twr > P==>engh BUS BW Twr = 21.50  cacheusd(kbytes)= 160.8   T4s(ms)= 8890.2398
mr  nc  nl  numrowun  num_loads
9   9   32    32         256.0  Twr > P==>engh BUS BW Twr = 23.90  cacheusd(kbytes)= 289.2   T4s(ms)= 8850.8286
mr  nc  nl  numrowun  num_loads
9   9   32    64         128.0  Twr > P==>engh BUS BW Twr = 25.31  cacheusd(kbytes)= 546.2   T4s(ms)= 8832.0446
mr  nc  nl  numrowun  num_loads
9   9   32    128         64.0  Twr > P==>engh BUS BW Twr = 26.08  cacheusd(kbytes)= 1060.2  T4s(ms)= 8824.4958
mr  nc  nl  numrowun  num_loads
9   9   32    256         32.0  Twr > P==>engh BUS BW Twr = 26.48  cacheusd(kbytes)= 2088.2  T4s(ms)= 8824.4078
mr  nc  nl  numrowun  num_loads
9   9   32    512         16.0  Twr > P==>engh BUS BW Twr = 26.69  cacheusd(kbytes)= 4144.2  T4s(ms)= 8831.7366
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   64    1         8192.0  Twr > P==>engh BUS BW Twr = 10.73  cacheusd(kbytes)= 40.3    T4s(ms)= 9298.4510
mr  nc  nl  numrowun  num_loads
9   9   64    2         4096.0  Twr > P==>engh BUS BW Twr = 17.89  cacheusd(kbytes)= 48.3    T4s(ms)= 8975.0910
mr  nc  nl  numrowun  num_loads
9   9   64    4         2048.0  Twr > P==>engh BUS BW Twr = 26.84  cacheusd(kbytes)= 64.4    T4s(ms)= 8813.4398
mr  nc  nl  numrowun  num_loads
9   9   64    8         1024.0  Twr > P==>engh BUS BW Twr = 35.81  cacheusd(kbytes)= 96.5    T4s(ms)= 8732.6718
mr  nc  nl  numrowun  num_loads
9   9   64    16         512.0  Twr > P==>engh BUS BW Twr = 43.00  cacheusd(kbytes)= 160.8   T4s(ms)= 8692.4030
mr  nc  nl  numrowun  num_loads
9   9   64    32         256.0  Twr > P==>engh BUS BW Twr = 47.79  cacheusd(kbytes)= 289.2   T4s(ms)= 8672.4990
mr  nc  nl  numrowun  num_loads
```

```
9   9   64   64          128.0  Twr > P==>engh BUS BW Twr = 50.62   cacheusd(kbytes)= 546.2    T4s(ms)= 8663.0078
mr  nc  nl  numrowun  num_loads
9   9   64   128          64.0  Twr > P==>engh BUS BW Twr = 52.16   cacheusd(kbytes)= 1060.2    T4s(ms)= 8659.1838
mr  nc  nl  numrowun  num_loads
9   9   64   256          32.0  Twr > P==>engh BUS BW Twr = 52.96   cacheusd(kbytes)= 2088.2    T4s(ms)= 8659.1150
mr  nc  nl  numrowun  num_loads
9   9   64   512          16.0  Twr > P==>engh BUS BW Twr = 53.37   cacheusd(kbytes)= 4144.2    T4s(ms)= 8662.7670
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   128   1          8192.0  Twr > P==>engh BUS BW Twr = 21.46   cacheusd(kbytes)= 40.3    T4s(ms)= 8902.4606
mr  nc  nl  numrowun  num_loads
9   9   128   2          4096.0  Twr > P==>engh BUS BW Twr = 35.76   cacheusd(kbytes)= 48.3    T4s(ms)= 8737.6062
mr  nc  nl  numrowun  num_loads
9   9   128   4          2048.0  Twr > P==>engh BUS BW Twr = 53.67   cacheusd(kbytes)= 64.4    T4s(ms)= 8655.1934
mr  nc  nl  numrowun  num_loads
9   9   128   8          1024.0  Twr > P==>engh BUS BW Twr = 71.61   cacheusd(kbytes)= 96.5    T4s(ms)= 8614.0158
mr  nc  nl  numrowun  num_loads
9   9   128   16          512.0  Twr > P==>engh BUS BW Twr = 85.99   cacheusd(kbytes)= 160.8    T4s(ms)= 8593.4846
mr  nc  nl  numrowun  num_loads
9   9   128   32          256.0  Twr > P==>engh BUS BW Twr = 95.58   cacheusd(kbytes)= 289.2    T4s(ms)= 8583.3342
mr  nc  nl  numrowun  num_loads
9   9   128   64          128.0  Twr > P==>engh BUS BW Twr = 101.23   cacheusd(kbytes)= 546.2    T4s(ms)= 8578.4894
mr  nc  nl  numrowun  num_loads
9   9   128   128          64.0  Twr > P==>engh BUS BW Twr = 104.31   cacheusd(kbytes)= 1060.2    T4s(ms)= 8576.5278
mr  nc  nl  numrowun  num_loads
9   9   128   256          32.0  Twr > P==>engh BUS BW Twr = 105.92   cacheusd(kbytes)= 2088.2    T4s(ms)= 8576.4686
mr  nc  nl  numrowun  num_loads
9   9   128   512          16.0  Twr > P==>engh BUS BW Twr = 106.75   cacheusd(kbytes)= 4144.2    T4s(ms)= 8578.2822
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256   1          8192.0  Twr > P==>engh BUS BW Twr = 42.84   cacheusd(kbytes)= 40.3    T4s(ms)= 8704.6190
mr  nc  nl  numrowun  num_loads
9   9   256   2          4096.0  Twr > P==>engh BUS BW Twr = 71.47   cacheusd(kbytes)= 48.3    T4s(ms)= 8618.8638
mr  nc  nl  numrowun  num_loads
9   9   256   4          2048.0  Twr > P==>engh BUS BW Twr = 107.27   cacheusd(kbytes)= 64.4    T4s(ms)= 8576.0702
mr  nc  nl  numrowun  num_loads
9   9   256   8          1024.0  Twr > P==>engh BUS BW Twr = 143.16   cacheusd(kbytes)= 96.5    T4s(ms)= 8554.6878
mr  nc  nl  numrowun  num_loads
9   9   256   16          512.0  Twr > P==>engh BUS BW Twr = 171.93   cacheusd(kbytes)= 160.8    T4s(ms)= 8544.0254
mr  nc  nl  numrowun  num_loads
9   9   256   32          256.0  Twr > P==>engh BUS BW Twr = 191.13   cacheusd(kbytes)= 289.2    T4s(ms)= 8538.7518
mr  nc  nl  numrowun  num_loads
9   9   256   64          128.0  Twr > P==>engh BUS BW Twr = 202.44   cacheusd(kbytes)= 546.2    T4s(ms)= 8536.2302
mr  nc  nl  numrowun  num_loads
9   9   256   128          64.0  Twr > P==>engh BUS BW Twr = 208.61   cacheusd(kbytes)= 1060.2    T4s(ms)= 8535.1998
mr  nc  nl  numrowun  num_loads
9   9   256   256          32.0  Twr > P==>engh BUS BW Twr = 211.84   cacheusd(kbytes)= 2088.2    T4s(ms)= 8535.1454
mr  nc  nl  numrowun  num_loads
9   9   256   512          16.0  Twr > P==>engh BUS BW Twr = 213.49   cacheusd(kbytes)= 4144.2    T4s(ms)= 8536.0398
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   512   1          8192.0  Twr > P==>engh BUS BW Twr = 85.37   cacheusd(kbytes)= 40.3    T4s(ms)= 8605.6982
mr  nc  nl  numrowun  num_loads
9   9   512   2          4096.0  Twr > P==>engh BUS BW Twr = 142.50   cacheusd(kbytes)= 48.3    T4s(ms)= 8559.5694
mr  nc  nl  numrowun  num_loads
9   9   512   4          2048.0  Twr > P==>engh BUS BW Twr = 214.27   cacheusd(kbytes)= 64.4    T4s(ms)= 8536.5086
mr  nc  nl  numrowun  num_loads
9   9   512   8          1024.0  Twr > P==>engh BUS BW Twr = 286.08   cacheusd(kbytes)= 96.5    T4s(ms)= 8525.0238
mr  nc  nl  numrowun  num_loads
9   9   512   16          512.0  Twr > P==>engh BUS BW Twr = 343.67   cacheusd(kbytes)= 160.8    T4s(ms)= 8519.2958
mr  nc  nl  numrowun  num_loads
9   9   512   32          256.0  Twr > P==>engh BUS BW Twr = 382.15   cacheusd(kbytes)= 289.2    T4s(ms)= 8516.4606
mr  nc  nl  numrowun  num_loads
9   9   512   64          128.0  Twr > P==>engh BUS BW Twr = 404.81   cacheusd(kbytes)= 546.2    T4s(ms)= 8515.1006
mr  nc  nl  numrowun  num_loads
9   9   512   128          64.0  Twr > P==>engh BUS BW Twr = 417.18   cacheusd(kbytes)= 1060.2    T4s(ms)= 8514.5358
mr  nc  nl  numrowun  num_loads
```

```
9   9  512  256          32.0 Twr > P==>engh BUS BW Twr = 423.66   cacheusd(kbytes)= 2088.2    T4s(ms)= 8514.4838
mr  nc  nl  numrowun  num_loads
9   9  512  512          16.0 Twr > P==>engh BUS BW Twr = 426.97   cacheusd(kbytes)= 4144.2    T4s(ms)= 8514.9186
*******************************************************************************
*******************************************************************************
```

P=4

```
mr  nc  nl  numrowun  num_loads
2   2   32   1          8192.0 Twr < P ==>btlnck  Twr = 0.9    cacheued(kbytes)= 12.0     T8s(ms)= 945.1528
mr  nc  nl  numrowun  num_loads
2   2   32   2          4096.0 Twr < P ==>btlnck  Twr = 1.1    cacheued(kbytes)= 20.0     T8s(ms)= 787.4568
mr  nc  nl  numrowun  num_loads
2   2   32   4          2048.0 Twr < P ==>btlnck  Twr = 1.2    cacheued(kbytes)= 36.0     T8s(ms)= 708.4552
mr  nc  nl  numrowun  num_loads
2   2   32   8          1024.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 68.0     T8s(ms)= 668.9544
mr  nc  nl  numrowun  num_loads
2   2   32   16          512.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 132.1    T8s(ms)= 649.2040
mr  nc  nl  numrowun  num_loads
2   2   32   32          256.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 260.1    T8s(ms)= 639.3288
mr  nc  nl  numrowun  num_loads
2   2   32   64          128.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 516.3    T8s(ms)= 634.3912
mr  nc  nl  numrowun  num_loads
2   2   32   128          64.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 1028.5   T8s(ms)= 631.9224
mr  nc  nl  numrowun  num_loads
2   2   32   256          32.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 2053.0   T8s(ms)= 630.6880
mr  nc  nl  numrowun  num_loads
2   2   32   512          16.0 Twr < P ==>btlnck  Twr = 1.3    cacheued(kbytes)= 4102.0   T8s(ms)= 630.0708
*******************************************************************************
mr  nc  nl  numrowun  num_loads
2   2   64   1          8192.0 Twr < P ==>btlnck  Twr = 1.8    cacheued(kbytes)= 12.0     T8s(ms)= 473.3845
mr  nc  nl  numrowun  num_loads
2   2   64   2          4096.0 Twr < P ==>btlnck  Twr = 2.1    cacheued(kbytes)= 20.0     T8s(ms)= 394.1142
mr  nc  nl  numrowun  num_loads
2   2   64   4          2048.0 Twr < P ==>btlnck  Twr = 2.4    cacheued(kbytes)= 36.0     T8s(ms)= 354.7673
mr  nc  nl  numrowun  num_loads
2   2   64   8          1024.0 Twr < P ==>btlnck  Twr = 2.5    cacheued(kbytes)= 68.0     T8s(ms)= 335.2863
mr  nc  nl  numrowun  num_loads
2   2   64   16          512.0 Twr < P ==>btlnck  Twr = 2.6    cacheued(kbytes)= 132.1    T8s(ms)= 326.0843
mr  nc  nl  numrowun  num_loads
2   2   64   32          256.0 Twr < P ==>btlnck  Twr = 2.6    cacheued(kbytes)= 260.1    T8s(ms)= 322.5603
mr  nc  nl  numrowun  num_loads
2   2   64   64          128.0 Twr < P ==>btlnck  Twr = 2.6    cacheued(kbytes)= 516.3    T8s(ms)= 322.9523
mr  nc  nl  numrowun  num_loads
2   2   64   128          64.0 Twr < P ==>btlnck  Twr = 2.7    cacheued(kbytes)= 1028.5   T8s(ms)= 327.4563
mr  nc  nl  numrowun  num_loads
2   2   64   256          32.0 Twr < P ==>btlnck  Twr = 2.7    cacheued(kbytes)= 2053.0   T8s(ms)= 338.3243
mr  nc  nl  numrowun  num_loads
2   2   64   512          16.0 Twr < P ==>btlnck  Twr = 2.7    cacheued(kbytes)= 4102.0   T8s(ms)= 360.9903
*******************************************************************************
mr  nc  nl  numrowun  num_loads
2   2  128   1          8192.0 Twr < P ==>btlnck  Twr = 3.6    cacheued(kbytes)= 12.0     T8s(ms)= 271.5106
mr  nc  nl  numrowun  num_loads
2   2  128   2          4096.0 Twr > P==>engh BUS BW Twr = 4.28   cacheued(kbytes)= 20.0     T8s(ms)= 260.4161
mr  nc  nl  numrowun  num_loads
2   2  128   4          2048.0 Twr > P==>engh BUS BW Twr = 4.75   cacheued(kbytes)= 36.0     T8s(ms)= 254.9569
mr  nc  nl  numrowun  num_loads
2   2  128   8          1024.0 Twr > P==>engh BUS BW Twr = 5.02   cacheued(kbytes)= 68.0     T8s(ms)= 252.3907
mr  nc  nl  numrowun  num_loads
2   2  128   16          512.0 Twr > P==>engh BUS BW Twr = 5.17   cacheued(kbytes)= 132.1    T8s(ms)= 251.3860
mr  nc  nl  numrowun  num_loads
2   2  128   32          256.0 Twr > P==>engh BUS BW Twr = 5.25   cacheued(kbytes)= 260.1    T8s(ms)= 251.4599
mr  nc  nl  numrowun  num_loads
2   2  128   64          128.0 Twr > P==>engh BUS BW Twr = 5.29   cacheued(kbytes)= 516.3    T8s(ms)= 252.6493
mr  nc  nl  numrowun  num_loads
2   2  128   128          64.0 Twr > P==>engh BUS BW Twr = 5.31   cacheued(kbytes)= 1028.5   T8s(ms)= 255.5489
mr  nc  nl  numrowun  num_loads
```

```
2   2   128   256          32.0  Twr > P==>engh BUS BW Twr = 5.32   cacheued(kbytes)= 2053.0     T8s(ms)= 261.6085
mr  nc  nl  numrowun  num_loads
2   2   128   512          16.0  Twr > P==>engh BUS BW Twr = 5.33   cacheued(kbytes)= 4102.0     T8s(ms)= 273.8579
*****************************************************************************

 mr  nc  nl  numrowun  num_loads
2   2   256   1          8192.0  Twr > P==>engh BUS BW Twr = 7.10   cacheued(kbytes)= 12.0      T8s(ms)= 241.9537
mr  nc  nl  numrowun  num_loads
2   2   256   2          4096.0  Twr > P==>engh BUS BW Twr = 8.52   cacheued(kbytes)= 20.0      T8s(ms)= 235.7825
mr  nc  nl  numrowun  num_loads
2   2   256   4          2048.0  Twr > P==>engh BUS BW Twr = 9.47   cacheued(kbytes)= 36.0      T8s(ms)= 232.7329
mr  nc  nl  numrowun  num_loads
2   2   256   8          1024.0  Twr > P==>engh BUS BW Twr = 10.04   cacheued(kbytes)= 68.0     T8s(ms)= 231.2801
mr  nc  nl  numrowun  num_loads
2   2   256   16          512.0  Twr > P==>engh BUS BW Twr = 10.34   cacheued(kbytes)= 132.1    T8s(ms)= 230.7075
mr  nc  nl  numrowun  num_loads
2   2   256   32          256.0  Twr > P==>engh BUS BW Twr = 10.50   cacheued(kbytes)= 260.1    T8s(ms)= 230.7044
mr  nc  nl  numrowun  num_loads
2   2   256   64          128.0  Twr > P==>engh BUS BW Twr = 10.59   cacheued(kbytes)= 516.3    T8s(ms)= 231.2791
mr  nc  nl  numrowun  num_loads
2   2   256   128          64.0  Twr > P==>engh BUS BW Twr = 10.63   cacheued(kbytes)= 1028.5    T8s(ms)= 232.7189
mr  nc  nl  numrowun  num_loads
2   2   256   256          32.0  Twr > P==>engh BUS BW Twr = 10.65   cacheued(kbytes)= 2053.0    T8s(ms)= 235.7437
mr  nc  nl  numrowun  num_loads
2   2   256   512          16.0  Twr > P==>engh BUS BW Twr = 10.66   cacheued(kbytes)= 4102.0    T8s(ms)= 241.8659
*****************************************************************************

 mr  nc  nl  numrowun  num_loads
2   2   512   1          8192.0  Twr > P==>engh BUS BW Twr = 14.04   cacheued(kbytes)= 12.0      T8s(ms)= 227.1913
mr  nc  nl  numrowun  num_loads
2   2   512   2          4096.0  Twr > P==>engh BUS BW Twr = 16.92   cacheued(kbytes)= 20.0      T8s(ms)= 223.4657
mr  nc  nl  numrowun  num_loads
2   2   512   4          2048.0  Twr > P==>engh BUS BW Twr = 18.87   cacheued(kbytes)= 36.0      T8s(ms)= 221.6209
mr  nc  nl  numrowun  num_loads
2   2   512   8          1024.0  Twr > P==>engh BUS BW Twr = 20.03   cacheued(kbytes)= 68.0      T8s(ms)= 220.7345
mr  nc  nl  numrowun  num_loads
2   2   512   16          512.0  Twr > P==>engh BUS BW Twr = 20.67   cacheued(kbytes)= 132.1     T8s(ms)= 220.3633
mr  nc  nl  numrowun  num_loads
2   2   512   32          256.0  Twr > P==>engh BUS BW Twr = 20.99   cacheued(kbytes)= 260.1     T8s(ms)= 220.3267
mr  nc  nl  numrowun  num_loads
2   2   512   64          128.0  Twr > P==>engh BUS BW Twr = 21.16   cacheued(kbytes)= 516.3     T8s(ms)= 220.5940
mr  nc  nl  numrowun  num_loads
2   2   512   128          64.0  Twr > P==>engh BUS BW Twr = 21.25   cacheued(kbytes)= 1028.5    T8s(ms)= 221.3039
mr  nc  nl  numrowun  num_loads
2   2   512   256          32.0  Twr > P==>engh BUS BW Twr = 21.29   cacheued(kbytes)= 2053.0    T8s(ms)= 222.8113
mr  nc  nl  numrowun  num_loads
2   2   512   512          16.0  Twr > P==>engh BUS BW Twr = 21.32   cacheued(kbytes)= 4102.0    T8s(ms)= 225.8699
*****************************************************************************
*****************************************************************************

 mr  nc  nl  numrowun  num_loads
3   3   32   1          8192.0  Twr < P ==>btlnck  Twr = 1.5   cacheued(kbytes)= 16.0     T8s(ms)= 1261.1863
mr  nc  nl  numrowun  num_loads
3   3   32   2          4096.0  Twr < P ==>btlnck  Twr = 2.0   cacheued(kbytes)= 24.0     T8s(ms)= 945.8201
mr  nc  nl  numrowun  num_loads
3   3   32   4          2048.0  Twr < P ==>btlnck  Twr = 2.4   cacheued(kbytes)= 40.0     T8s(ms)= 788.4828
mr  nc  nl  numrowun  num_loads
3   3   32   8          1024.0  Twr < P ==>btlnck  Twr = 2.7   cacheued(kbytes)= 72.1     T8s(ms)= 710.5058
mr  nc  nl  numrowun  num_loads
3   3   32   16          512.0  Twr < P ==>btlnck  Twr = 2.8   cacheued(kbytes)= 136.1    T8s(ms)= 672.9006
mr  nc  nl  numrowun  num_loads
3   3   32   32          256.0  Twr < P ==>btlnck  Twr = 2.9   cacheued(kbytes)= 264.3    T8s(ms)= 656.8646
mr  nc  nl  numrowun  num_loads
3   3   32   64          128.0  Twr < P ==>btlnck  Twr = 3.0   cacheued(kbytes)= 520.5    T8s(ms)= 654.3798
mr  nc  nl  numrowun  num_loads
3   3   32   128          64.0  Twr < P ==>btlnck  Twr = 3.0   cacheued(kbytes)= 1033.0    T8s(ms)= 664.2038
mr  nc  nl  numrowun  num_loads
3   3   32   256          32.0  Twr < P ==>btlnck  Twr = 3.0   cacheued(kbytes)= 2058.0    T8s(ms)= 691.2486
mr  nc  nl  numrowun  num_loads
3   3   32   512          16.0  Twr < P ==>btlnck  Twr = 3.0   cacheued(kbytes)= 4108.0    T8s(ms)= 749.0366
```

```
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   64    1         8192.0  Twr < P ==>btlnck  Twr = 3.0    cacheued(kbytes)= 16.0     T8s(ms)= 632.6152
mr  nc  nl  numrowun  num_loads
3   3   64    2         4096.0  Twr > P==>engh BUS BW Twr = 4.00   cacheued(kbytes)= 24.0     T8s(ms)= 591.8087
mr  nc  nl  numrowun  num_loads
3   3   64    4         2048.0  Twr > P==>engh BUS BW Twr = 4.80   cacheued(kbytes)= 40.0     T8s(ms)= 571.5588
mr  nc  nl  numrowun  num_loads
3   3   64    8         1024.0  Twr > P==>engh BUS BW Twr = 5.33   cacheued(kbytes)= 72.1     T8s(ms)= 561.7607
mr  nc  nl  numrowun  num_loads
3   3   64   16          512.0  Twr > P==>engh BUS BW Twr = 5.64   cacheued(kbytes)= 136.1    T8s(ms)= 557.4381
mr  nc  nl  numrowun  num_loads
3   3   64   32          256.0  Twr > P==>engh BUS BW Twr = 5.81   cacheued(kbytes)= 264.3    T8s(ms)= 556.4297
mr  nc  nl  numrowun  num_loads
3   3   64   64          128.0  Twr > P==>engh BUS BW Twr = 5.90   cacheued(kbytes)= 520.5    T8s(ms)= 558.2313
mr  nc  nl  numrowun  num_loads
3   3   64  128           64.0  Twr > P==>engh BUS BW Twr = 5.95   cacheued(kbytes)= 1033.0    T8s(ms)= 563.7437
mr  nc  nl  numrowun  num_loads
3   3   64  256           32.0  Twr > P==>engh BUS BW Twr = 5.97   cacheued(kbytes)= 2058.0    T8s(ms)= 575.7231
mr  nc  nl  numrowun  num_loads
3   3   64  512           16.0  Twr > P==>engh BUS BW Twr = 5.98   cacheued(kbytes)= 4108.0    T8s(ms)= 600.1592
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  128    1         8192.0  Twr > P==>engh BUS BW Twr = 6.01   cacheued(kbytes)= 16.0     T8s(ms)= 553.5585
mr  nc  nl  numrowun  num_loads
3   3  128    2         4096.0  Twr > P==>engh BUS BW Twr = 8.00   cacheued(kbytes)= 24.0     T8s(ms)= 532.5377
mr  nc  nl  numrowun  num_loads
3   3  128    4         2048.0  Twr > P==>engh BUS BW Twr = 9.59   cacheued(kbytes)= 40.0     T8s(ms)= 522.1379
mr  nc  nl  numrowun  num_loads
3   3  128    8         1024.0  Twr > P==>engh BUS BW Twr = 10.66   cacheued(kbytes)= 72.1     T8s(ms)= 517.0628
mr  nc  nl  numrowun  num_loads
3   3  128   16          512.0  Twr > P==>engh BUS BW Twr = 11.29   cacheued(kbytes)= 136.1    T8s(ms)= 514.8135
mr  nc  nl  numrowun  num_loads
3   3  128   32          256.0  Twr > P==>engh BUS BW Twr = 11.63   cacheued(kbytes)= 264.3    T8s(ms)= 514.2653
mr  nc  nl  numrowun  num_loads
3   3  128   64          128.0  Twr > P==>engh BUS BW Twr = 11.81   cacheued(kbytes)= 520.5    T8s(ms)= 515.1441
mr  nc  nl  numrowun  num_loads
3   3  128  128           64.0  Twr > P==>engh BUS BW Twr = 11.90   cacheued(kbytes)= 1033.0    T8s(ms)= 517.8893
mr  nc  nl  numrowun  num_loads
3   3  128  256           32.0  Twr > P==>engh BUS BW Twr = 11.95   cacheued(kbytes)= 2058.0    T8s(ms)= 523.8735
mr  nc  nl  numrowun  num_loads
3   3  128  512           16.0  Twr > P==>engh BUS BW Twr = 11.97   cacheued(kbytes)= 4108.0    T8s(ms)= 536.0888
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  256    1         8192.0  Twr > P==>engh BUS BW Twr = 11.96   cacheued(kbytes)= 16.0     T8s(ms)= 514.1937
mr  nc  nl  numrowun  num_loads
3   3  256    2         4096.0  Twr > P==>engh BUS BW Twr = 15.95   cacheued(kbytes)= 24.0     T8s(ms)= 502.9793
mr  nc  nl  numrowun  num_loads
3   3  256    4         2048.0  Twr > P==>engh BUS BW Twr = 19.16   cacheued(kbytes)= 40.0     T8s(ms)= 497.4081
mr  nc  nl  numrowun  num_loads
3   3  256    8         1024.0  Twr > P==>engh BUS BW Twr = 21.29   cacheued(kbytes)= 72.1     T8s(ms)= 494.7139
mr  nc  nl  numrowun  num_loads
3   3  256   16          512.0  Twr > P==>engh BUS BW Twr = 22.56   cacheued(kbytes)= 136.1    T8s(ms)= 493.5012
mr  nc  nl  numrowun  num_loads
3   3  256   32          256.0  Twr > P==>engh BUS BW Twr = 23.25   cacheued(kbytes)= 264.3    T8s(ms)= 493.1831
mr  nc  nl  numrowun  num_loads
3   3  256   64          128.0  Twr > P==>engh BUS BW Twr = 23.61   cacheued(kbytes)= 520.5    T8s(ms)= 493.6005
mr  nc  nl  numrowun  num_loads
3   3  256  128           64.0  Twr > P==>engh BUS BW Twr = 23.80   cacheued(kbytes)= 1033.0    T8s(ms)= 494.9621
mr  nc  nl  numrowun  num_loads
3   3  256  256           32.0  Twr > P==>engh BUS BW Twr = 23.89   cacheued(kbytes)= 2058.0    T8s(ms)= 497.9487
mr  nc  nl  numrowun  num_loads
3   3  256  512           16.0  Twr > P==>engh BUS BW Twr = 23.94   cacheued(kbytes)= 4108.0    T8s(ms)= 504.0536
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  512    1         8192.0  Twr > P==>engh BUS BW Twr = 23.70   cacheued(kbytes)= 16.0     T8s(ms)= 494.5113
mr  nc  nl  numrowun  num_loads
```

```
3   3   512   2          4096.0  Twr > P==>engh BUS BW Twr = 31.71   cacheued(kbytes)= 24.0     T8s(ms)= 488.2001
mr  nc  nl  numrowun  num_loads
3   3   512   4          2048.0  Twr > P==>engh BUS BW Twr = 38.18   cacheued(kbytes)= 40.0     T8s(ms)= 485.0625
mr  nc  nl  numrowun  num_loads
3   3   512   8          1024.0  Twr > P==>engh BUS BW Twr = 42.54   cacheued(kbytes)= 72.1     T8s(ms)= 483.5297
mr  nc  nl  numrowun  num_loads
3   3   512   16          512.0  Twr > P==>engh BUS BW Twr = 45.07   cacheued(kbytes)= 136.1    T8s(ms)= 482.8451
mr  nc  nl  numrowun  num_loads
3   3   512   32          256.0  Twr > P==>engh BUS BW Twr = 46.47   cacheued(kbytes)= 264.3    T8s(ms)= 482.6420
mr  nc  nl  numrowun  num_loads
3   3   512   64          128.0  Twr > P==>engh BUS BW Twr = 47.21   cacheued(kbytes)= 520.5    T8s(ms)= 482.8287
mr  nc  nl  numrowun  num_loads
3   3   512   128          64.0  Twr > P==>engh BUS BW Twr = 47.59   cacheued(kbytes)= 1033.0   T8s(ms)= 483.4985
mr  nc  nl  numrowun  num_loads
3   3   512   256          32.0  Twr > P==>engh BUS BW Twr = 47.78   cacheued(kbytes)= 2058.0   T8s(ms)= 484.9863
mr  nc  nl  numrowun  num_loads
3   3   512   512          16.0  Twr > P==>engh BUS BW Twr = 47.87   cacheued(kbytes)= 4108.0   T8s(ms)= 488.0360
*********************************************************************************
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6   32    1          8192.0  Twr < P ==>btlnck  Twr = 3.4    cacheued(kbytes)= 28.1     T8s(ms)= 2445.7177
mr  nc  nl  numrowun  num_loads
6   6   32    2          4096.0  Twr > P==>engh BUS BW Twr = 5.32   cacheued(kbytes)= 36.1     T8s(ms)= 2245.7068
mr  nc  nl  numrowun  num_loads
6   6   32    4          2048.0  Twr > P==>engh BUS BW Twr = 7.36   cacheued(kbytes)= 52.2     T8s(ms)= 2146.2306
mr  nc  nl  numrowun  num_loads
6   6   32    8          1024.0  Twr > P==>engh BUS BW Twr = 9.12   cacheued(kbytes)= 84.3     T8s(ms)= 2097.0929
mr  nc  nl  numrowun  num_loads
6   6   32    16          512.0  Twr > P==>engh BUS BW Twr = 10.35   cacheued(kbytes)= 148.4    T8s(ms)= 2073.6783
mr  nc  nl  numrowun  num_loads
6   6   32    32          256.0  Twr > P==>engh BUS BW Twr = 11.10   cacheued(kbytes)= 276.7    T8s(ms)= 2064.2795
mr  nc  nl  numrowun  num_loads
6   6   32    64          128.0  Twr > P==>engh BUS BW Twr = 11.52   cacheued(kbytes)= 533.3    T8s(ms)= 2064.1971
mr  nc  nl  numrowun  num_loads
6   6   32    128          64.0  Twr > P==>engh BUS BW Twr = 11.74   cacheued(kbytes)= 1046.6   T8s(ms)= 2073.3899
mr  nc  nl  numrowun  num_loads
6   6   32    256          32.0  Twr > P==>engh BUS BW Twr = 11.86   cacheued(kbytes)= 2073.1   T8s(ms)= 2096.4543
mr  nc  nl  numrowun  num_loads
6   6   32    512          16.0  Twr > P==>engh BUS BW Twr = 11.91   cacheued(kbytes)= 4126.1   T8s(ms)= 2144.9225
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6   64    1          8192.0  Twr > P==>engh BUS BW Twr = 6.84   cacheued(kbytes)= 28.1     T8s(ms)= 2168.4044
mr  nc  nl  numrowun  num_loads
6   6   64    2          4096.0  Twr > P==>engh BUS BW Twr = 10.63   cacheued(kbytes)= 36.1     T8s(ms)= 2067.6365
mr  nc  nl  numrowun  num_loads
6   6   64    4          2048.0  Twr > P==>engh BUS BW Twr = 14.72   cacheued(kbytes)= 52.2     T8s(ms)= 2017.4354
mr  nc  nl  numrowun  num_loads
6   6   64    8          1024.0  Twr > P==>engh BUS BW Twr = 18.23   cacheued(kbytes)= 84.3     T8s(ms)= 1992.6041
mr  nc  nl  numrowun  num_loads
6   6   64    16          512.0  Twr > P==>engh BUS BW Twr = 20.70   cacheued(kbytes)= 148.4    T8s(ms)= 1980.7656
mr  nc  nl  numrowun  num_loads
6   6   64    32          256.0  Twr > P==>engh BUS BW Twr = 22.20   cacheued(kbytes)= 276.7    T8s(ms)= 1976.0006
mr  nc  nl  numrowun  num_loads
6   6   64    64          128.0  Twr > P==>engh BUS BW Twr = 23.04   cacheued(kbytes)= 533.3    T8s(ms)= 1975.9266
mr  nc  nl  numrowun  num_loads
6   6   64    128          64.0  Twr > P==>engh BUS BW Twr = 23.48   cacheued(kbytes)= 1046.6   T8s(ms)= 1980.5066
mr  nc  nl  numrowun  num_loads
6   6   64    256          32.0  Twr > P==>engh BUS BW Twr = 23.71   cacheued(kbytes)= 2073.1   T8s(ms)= 1992.0306
mr  nc  nl  numrowun  num_loads
6   6   64    512          16.0  Twr > P==>engh BUS BW Twr = 23.83   cacheued(kbytes)= 4126.1   T8s(ms)= 2016.2606
*********************************************************************************
mr  nc  nl  numrowun  num_loads
6   6   128   1          8192.0  Twr > P==>engh BUS BW Twr = 13.66   cacheued(kbytes)= 28.1     T8s(ms)= 2030.0198
mr  nc  nl  numrowun  num_loads
6   6   128   2          4096.0  Twr > P==>engh BUS BW Twr = 21.24   cacheued(kbytes)= 36.1     T8s(ms)= 1978.6631
mr  nc  nl  numrowun  num_loads
6   6   128   4          2048.0  Twr > P==>engh BUS BW Twr = 29.43   cacheued(kbytes)= 52.2     T8s(ms)= 1953.0185
```

```
mr  nc  nl  numrowun  num_loads
6   6   128   8        1024.0  Twr > P==>engh BUS BW Twr = 36.44   cacheued(kbytes)= 84.3     T8s(ms)= 1940.3597
mr  nc  nl  numrowun  num_loads
6   6   128   16        512.0  Twr > P==>engh BUS BW Twr = 41.39   cacheued(kbytes)= 148.4    T8s(ms)= 1934.3093
mr  nc  nl  numrowun  num_loads
6   6   128   32        256.0  Twr > P==>engh BUS BW Twr = 44.40   cacheued(kbytes)= 276.7    T8s(ms)= 1931.8612
mr  nc  nl  numrowun  num_loads
6   6   128   64        128.0  Twr > P==>engh BUS BW Twr = 46.08   cacheued(kbytes)= 533.3    T8s(ms)= 1931.7914
mr  nc  nl  numrowun  num_loads
6   6   128   128        64.0  Twr > P==>engh BUS BW Twr = 46.97   cacheued(kbytes)= 1046.6   T8s(ms)= 1934.0650
mr  nc  nl  numrowun  num_loads
6   6   128   256        32.0  Twr > P==>engh BUS BW Twr = 47.42   cacheued(kbytes)= 2073.1   T8s(ms)= 1939.8188
mr  nc  nl  numrowun  num_loads
6   6   128   512        16.0  Twr > P==>engh BUS BW Twr = 47.65   cacheued(kbytes)= 4126.1   T8s(ms)= 1951.9297
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   256   1        8192.0  Twr > P==>engh BUS BW Twr = 27.28   cacheued(kbytes)= 28.1     T8s(ms)= 1960.8275
mr  nc  nl  numrowun  num_loads
6   6   256   2        4096.0  Twr > P==>engh BUS BW Twr = 42.39   cacheued(kbytes)= 36.1     T8s(ms)= 1934.1764
mr  nc  nl  numrowun  num_loads
6   6   256   4        2048.0  Twr > P==>engh BUS BW Twr = 58.81   cacheued(kbytes)= 52.2     T8s(ms)= 1920.8100
mr  nc  nl  numrowun  num_loads
6   6   256   8        1024.0  Twr > P==>engh BUS BW Twr = 72.82   cacheued(kbytes)= 84.3     T8s(ms)= 1914.2375
mr  nc  nl  numrowun  num_loads
6   6   256   16        512.0  Twr > P==>engh BUS BW Twr = 82.75   cacheued(kbytes)= 148.4    T8s(ms)= 1911.0762
mr  nc  nl  numrowun  num_loads
6   6   256   32        256.0  Twr > P==>engh BUS BW Twr = 88.79   cacheued(kbytes)= 276.7    T8s(ms)= 1909.7890
mr  nc  nl  numrowun  num_loads
6   6   256   64        128.0  Twr > P==>engh BUS BW Twr = 92.15   cacheued(kbytes)= 533.3    T8s(ms)= 1909.7225
mr  nc  nl  numrowun  num_loads
6   6   256   128        64.0  Twr > P==>engh BUS BW Twr = 93.93   cacheued(kbytes)= 1046.6   T8s(ms)= 1910.8435
mr  nc  nl  numrowun  num_loads
6   6   256   256        32.0  Twr > P==>engh BUS BW Twr = 94.84   cacheued(kbytes)= 2073.1   T8s(ms)= 1913.7125
mr  nc  nl  numrowun  num_loads
6   6   256   512        16.0  Twr > P==>engh BUS BW Twr = 95.31   cacheued(kbytes)= 4126.1   T8s(ms)= 1919.7640
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   512   1        8192.0  Twr > P==>engh BUS BW Twr = 54.41   cacheued(kbytes)= 28.1     T8s(ms)= 1926.2313
mr  nc  nl  numrowun  num_loads
6   6   512   2        4096.0  Twr > P==>engh BUS BW Twr = 84.45   cacheued(kbytes)= 36.1     T8s(ms)= 1911.9331
mr  nc  nl  numrowun  num_loads
6   6   512   4        2048.0  Twr > P==>engh BUS BW Twr = 117.28   cacheued(kbytes)= 52.2    T8s(ms)= 1904.7251
mr  nc  nl  numrowun  num_loads
6   6   512   8        1024.0  Twr > P==>engh BUS BW Twr = 145.40   cacheued(kbytes)= 84.3    T8s(ms)= 1901.1764
mr  nc  nl  numrowun  num_loads
6   6   512   16        512.0  Twr > P==>engh BUS BW Twr = 165.34   cacheued(kbytes)= 148.4    T8s(ms)= 1899.4646
mr  nc  nl  numrowun  num_loads
6   6   512   32        256.0  Twr > P==>engh BUS BW Twr = 177.52   cacheued(kbytes)= 276.7    T8s(ms)= 1898.7529
mr  nc  nl  numrowun  num_loads
6   6   512   64        128.0  Twr > P==>engh BUS BW Twr = 184.27   cacheued(kbytes)= 533.3    T8s(ms)= 1898.6880
mr  nc  nl  numrowun  num_loads
6   6   512   128        64.0  Twr > P==>engh BUS BW Twr = 187.84   cacheued(kbytes)= 1046.6   T8s(ms)= 1899.2327
mr  nc  nl  numrowun  num_loads
6   6   512   256        32.0  Twr > P==>engh BUS BW Twr = 189.67   cacheued(kbytes)= 2073.1   T8s(ms)= 1900.6593
mr  nc  nl  numrowun  num_loads
6   6   512   512        16.0  Twr > P==>engh BUS BW Twr = 190.61   cacheued(kbytes)= 4126.1   T8s(ms)= 1903.6811
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   32   1        8192.0  Twr > P==>engh BUS BW Twr = 5.37   cacheued(kbytes)= 40.3     T8s(ms)= 5046.3707
mr  nc  nl  numrowun  num_loads
9   9   32   2        4096.0  Twr > P==>engh BUS BW Twr = 8.95   cacheued(kbytes)= 48.3     T8s(ms)= 4725.9964
mr  nc  nl  numrowun  num_loads
9   9   32   4        2048.0  Twr > P==>engh BUS BW Twr = 13.42   cacheued(kbytes)= 64.4    T8s(ms)= 4566.2792
mr  nc  nl  numrowun  num_loads
9   9   32   8        1024.0  Twr > P==>engh BUS BW Twr = 17.91   cacheued(kbytes)= 96.5    T8s(ms)= 4486.9984
mr  nc  nl  numrowun  num_loads
```

```
9   9   32   16          512.0  Twr > P==>engh BUS BW Twr = 21.50   cacheued(kbytes)= 160.8     T8s(ms)= 4448.5136
mr  nc  nl  numrowun  num_loads
9   9   32   32          256.0  Twr > P==>engh BUS BW Twr = 23.90   cacheued(kbytes)= 289.2     T8s(ms)= 4431.5824
mr  nc  nl  numrowun  num_loads
9   9   32   64          128.0  Twr > P==>engh BUS BW Twr = 25.31   cacheued(kbytes)= 546.2     T8s(ms)= 4427.7392
mr  nc  nl  numrowun  num_loads
9   9   32   128          64.0  Twr > P==>engh BUS BW Twr = 26.08   cacheued(kbytes)= 1060.2    T8s(ms)= 4435.0624
mr  nc  nl  numrowun  num_loads
9   9   32   256          32.0  Twr > P==>engh BUS BW Twr = 26.48   cacheued(kbytes)= 2088.2    T8s(ms)= 4457.2136
mr  nc  nl  numrowun  num_loads
9   9   32   512          16.0  Twr > P==>engh BUS BW Twr = 26.69   cacheued(kbytes)= 4144.2    T8s(ms)= 4505.2684
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   64    1          8192.0  Twr > P==>engh BUS BW Twr = 10.73   cacheued(kbytes)= 40.3      T8s(ms)= 4649.6219
mr  nc  nl  numrowun  num_loads
9   9   64    2          4096.0  Twr > P==>engh BUS BW Twr = 17.89   cacheued(kbytes)= 48.3      T8s(ms)= 4488.0286
mr  nc  nl  numrowun  num_loads
9   9   64    4          2048.0  Twr > P==>engh BUS BW Twr = 26.84   cacheued(kbytes)= 64.4      T8s(ms)= 4407.3764
mr  nc  nl  numrowun  num_loads
9   9   64    8          1024.0  Twr > P==>engh BUS BW Twr = 35.81   cacheued(kbytes)= 96.5      T8s(ms)= 4367.3392
mr  nc  nl  numrowun  num_loads
9   9   64    16          512.0  Twr > P==>engh BUS BW Twr = 43.00   cacheued(kbytes)= 160.8     T8s(ms)= 4347.8984
mr  nc  nl  numrowun  num_loads
9   9   64    32          256.0  Twr > P==>engh BUS BW Twr = 47.79   cacheued(kbytes)= 289.2     T8s(ms)= 4339.3336
mr  nc  nl  numrowun  num_loads
9   9   64    64          128.0  Twr > P==>engh BUS BW Twr = 50.62   cacheued(kbytes)= 546.2     T8s(ms)= 4337.3624
mr  nc  nl  numrowun  num_loads
9   9   64    128          64.0  Twr > P==>engh BUS BW Twr = 52.16   cacheued(kbytes)= 1060.2    T8s(ms)= 4340.9992
mr  nc  nl  numrowun  num_loads
9   9   64    256          32.0  Twr > P==>engh BUS BW Twr = 52.96   cacheued(kbytes)= 2088.2    T8s(ms)= 4352.0624
mr  nc  nl  numrowun  num_loads
9   9   64    512          16.0  Twr > P==>engh BUS BW Twr = 53.37   cacheued(kbytes)= 4144.2    T8s(ms)= 4376.0836
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   128   1          8192.0  Twr > P==>engh BUS BW Twr = 21.46   cacheued(kbytes)= 40.3      T8s(ms)= 4451.4285
mr  nc  nl  numrowun  num_loads
9   9   128   2          4096.0  Twr > P==>engh BUS BW Twr = 35.76   cacheued(kbytes)= 48.3      T8s(ms)= 4369.0447
mr  nc  nl  numrowun  num_loads
9   9   128   4          2048.0  Twr > P==>engh BUS BW Twr = 53.67   cacheued(kbytes)= 64.4      T8s(ms)= 4327.9250
mr  nc  nl  numrowun  num_loads
9   9   128   8          1024.0  Twr > P==>engh BUS BW Twr = 71.61   cacheued(kbytes)= 96.5      T8s(ms)= 4307.5096
mr  nc  nl  numrowun  num_loads
9   9   128   16          512.0  Twr > P==>engh BUS BW Twr = 85.99   cacheued(kbytes)= 160.8     T8s(ms)= 4297.5908
mr  nc  nl  numrowun  num_loads
9   9   128   32          256.0  Twr > P==>engh BUS BW Twr = 95.58   cacheued(kbytes)= 289.2     T8s(ms)= 4293.2092
mr  nc  nl  numrowun  num_loads
9   9   128   64          128.0  Twr > P==>engh BUS BW Twr = 101.23   cacheued(kbytes)= 546.2     T8s(ms)= 4292.1740
mr  nc  nl  numrowun  num_loads
9   9   128   128          64.0  Twr > P==>engh BUS BW Twr = 104.31   cacheued(kbytes)= 1060.2    T8s(ms)= 4293.9676
mr  nc  nl  numrowun  num_loads
9   9   128   256          32.0  Twr > P==>engh BUS BW Twr = 105.92   cacheued(kbytes)= 2088.2    T8s(ms)= 4299.4868
mr  nc  nl  numrowun  num_loads
9   9   128   512          16.0  Twr > P==>engh BUS BW Twr = 106.75   cacheued(kbytes)= 4144.2    T8s(ms)= 4311.4912
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256   1          8192.0  Twr > P==>engh BUS BW Twr = 42.84   cacheued(kbytes)= 40.3      T8s(ms)= 4352.4087
mr  nc  nl  numrowun  num_loads
9   9   256   2          4096.0  Twr > P==>engh BUS BW Twr = 71.47   cacheued(kbytes)= 48.3      T8s(ms)= 4309.5527
mr  nc  nl  numrowun  num_loads
9   9   256   4          2048.0  Twr > P==>engh BUS BW Twr = 107.27   cacheued(kbytes)= 64.4      T8s(ms)= 4288.1993
mr  nc  nl  numrowun  num_loads
9   9   256   8          1024.0  Twr > P==>engh BUS BW Twr = 143.16   cacheued(kbytes)= 96.5      T8s(ms)= 4277.5948
mr  nc  nl  numrowun  num_loads
9   9   256   16          512.0  Twr > P==>engh BUS BW Twr = 171.93   cacheued(kbytes)= 160.8     T8s(ms)= 4272.4370
mr  nc  nl  numrowun  num_loads
9   9   256   32          256.0  Twr > P==>engh BUS BW Twr = 191.13   cacheued(kbytes)= 289.2     T8s(ms)= 4270.1470
mr  nc  nl  numrowun  num_loads
```

```
9   9  256   64        128.0 Twr > P==>engh BUS BW Twr = 202.44   cacheued(kbytes)= 546.2     T8s(ms)= 4269.5798
mr  nc  nl  numrowun  num_loads
9   9  256  128         64.0 Twr > P==>engh BUS BW Twr = 208.61   cacheued(kbytes)= 1060.2    T8s(ms)= 4270.4518
mr  nc  nl  numrowun  num_loads
9   9  256  256         32.0 Twr > P==>engh BUS BW Twr = 211.84   cacheued(kbytes)= 2088.2    T8s(ms)= 4273.1990
mr  nc  nl  numrowun  num_loads
9   9  256  512         16.0 Twr > P==>engh BUS BW Twr = 213.49   cacheued(kbytes)= 4144.2    T8s(ms)= 4279.1950
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9  512    1       8192.0 Twr > P==>engh BUS BW Twr = 85.37    cacheued(kbytes)= 40.3      T8s(ms)= 4302.8988
mr  nc  nl  numrowun  num_loads
9   9  512    2       4096.0 Twr > P==>engh BUS BW Twr = 142.50   cacheued(kbytes)= 48.3      T8s(ms)= 4279.8452
mr  nc  nl  numrowun  num_loads
9   9  512    4       2048.0 Twr > P==>engh BUS BW Twr = 214.27   cacheued(kbytes)= 64.4      T8s(ms)= 4268.3364
mr  nc  nl  numrowun  num_loads
9   9  512    8       1024.0 Twr > P==>engh BUS BW Twr = 286.08   cacheued(kbytes)= 96.5      T8s(ms)= 4262.6374
mr  nc  nl  numrowun  num_loads
9   9  512   16        512.0 Twr > P==>engh BUS BW Twr = 343.67   cacheued(kbytes)= 160.8     T8s(ms)= 4259.8601
mr  nc  nl  numrowun  num_loads
9   9  512   32        256.0 Twr > P==>engh BUS BW Twr = 382.15   cacheued(kbytes)= 289.2     T8s(ms)= 4258.6159
mr  nc  nl  numrowun  num_loads
9   9  512   64        128.0 Twr > P==>engh BUS BW Twr = 404.81   cacheued(kbytes)= 546.2     T8s(ms)= 4258.2827
mr  nc  nl  numrowun  num_loads
9   9  512  128         64.0 Twr > P==>engh BUS BW Twr = 417.18   cacheued(kbytes)= 1060.2    T8s(ms)= 4258.6939
mr  nc  nl  numrowun  num_loads
9   9  512  256         32.0 Twr > P==>engh BUS BW Twr = 423.66   cacheued(kbytes)= 2088.2    T8s(ms)= 4260.0551
mr  nc  nl  numrowun  num_loads
9   9  512  512         16.0 Twr > P==>engh BUS BW Twr = 426.97   cacheued(kbytes)= 4144.2    T8s(ms)= 4263.0469
*********************************************************************************
 *********************************************************************************
```

P=8

```
mr  nc  nl  numrowun  num_loads
2   2   32    1       8192.0 Twr < P ==>btlnck  Twr = 0.9    cacheusd(kbytes)= 12.0     T16s(ms)= 945.1536
mr  nc  nl  numrowun  num_loads
2   2   32    2       4096.0 Twr < P ==>btlnck  Twr = 1.1    cacheusd(kbytes)= 20.0     T16s(ms)= 787.4576
mr  nc  nl  numrowun  num_loads
2   2   32    4       2048.0 Twr < P ==>btlnck  Twr = 1.2    cacheusd(kbytes)= 36.0     T16s(ms)= 708.4560
mr  nc  nl  numrowun  num_loads
2   2   32    8       1024.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 68.0     T16s(ms)= 668.9552
mr  nc  nl  numrowun  num_loads
2   2   32   16        512.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 132.1    T16s(ms)= 649.2048
mr  nc  nl  numrowun  num_loads
2   2   32   32        256.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 260.1    T16s(ms)= 639.3296
mr  nc  nl  numrowun  num_loads
2   2   32   64        128.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 516.3    T16s(ms)= 634.3920
mr  nc  nl  numrowun  num_loads
2   2   32  128         64.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 1028.5   T16s(ms)= 631.9232
mr  nc  nl  numrowun  num_loads
2   2   32  256         32.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 2053.0   T16s(ms)= 630.6888
mr  nc  nl  numrowun  num_loads
2   2   32  512         16.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 4102.0   T16s(ms)= 630.0716
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   64    1       8192.0 Twr < P ==>btlnck  Twr = 1.8    cacheusd(kbytes)= 12.0     T16s(ms)= 473.2944
mr  nc  nl  numrowun  num_loads
2   2   64    2       4096.0 Twr < P ==>btlnck  Twr = 2.1    cacheusd(kbytes)= 20.0     T16s(ms)= 393.9344
mr  nc  nl  numrowun  num_loads
2   2   64    4       2048.0 Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 36.0     T16s(ms)= 354.4080
mr  nc  nl  numrowun  num_loads
2   2   64    8       1024.0 Twr < P ==>btlnck  Twr = 2.5    cacheusd(kbytes)= 68.0     T16s(ms)= 334.5680
mr  nc  nl  numrowun  num_loads
2   2   64   16        512.0 Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 132.1    T16s(ms)= 324.6480
mr  nc  nl  numrowun  num_loads
2   2   64   32        256.0 Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 260.1    T16s(ms)= 319.6880
mr  nc  nl  numrowun  num_loads
```

```
2   2   64   64          128.0 Twr < P ==>btlnck Twr = 2.6   cacheusd(kbytes)= 516.3    T16s(ms)= 317.2080
mr  nc  nl  numrowun  num_loads
2   2   64   128          64.0 Twr < P ==>btlnck Twr = 2.7   cacheusd(kbytes)= 1028.5    T16s(ms)= 315.9680
mr  nc  nl  numrowun  num_loads
2   2   64   256          32.0 Twr < P ==>btlnck Twr = 2.7   cacheusd(kbytes)= 2053.0    T16s(ms)= 315.3480
mr  nc  nl  numrowun  num_loads
2   2   64   512          16.0 Twr < P ==>btlnck Twr = 2.7   cacheusd(kbytes)= 4102.0    T16s(ms)= 315.0380
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   128   1          8192.0 Twr < P ==>btlnck Twr = 3.6   cacheusd(kbytes)= 12.0    T16s(ms)= 237.4357
mr  nc  nl  numrowun  num_loads
2   2   128   2          4096.0 Twr < P ==>btlnck Twr = 4.3   cacheusd(kbytes)= 20.0    T16s(ms)= 197.4678
mr  nc  nl  numrowun  num_loads
2   2   128   4          2048.0 Twr < P ==>btlnck Twr = 4.7   cacheusd(kbytes)= 36.0    T16s(ms)= 177.5897
mr  nc  nl  numrowun  num_loads
2   2   128   8          1024.0 Twr < P ==>btlnck Twr = 5.0   cacheusd(kbytes)= 68.0    T16s(ms)= 167.9391
mr  nc  nl  numrowun  num_loads
2   2   128   16          512.0 Twr < P ==>btlnck Twr = 5.2   cacheusd(kbytes)= 132.1    T16s(ms)= 163.4987
mr  nc  nl  numrowun  num_loads
2   2   128   32          256.0 Twr < P ==>btlnck Twr = 5.3   cacheusd(kbytes)= 260.1    T16s(ms)= 162.1251
mr  nc  nl  numrowun  num_loads
2   2   128   64          128.0 Twr < P ==>btlnck Twr = 5.3   cacheusd(kbytes)= 516.3    T16s(ms)= 163.1315
mr  nc  nl  numrowun  num_loads
2   2   128   128          64.0 Twr < P ==>btlnck Twr = 5.3   cacheusd(kbytes)= 1028.5    T16s(ms)= 167.0211
mr  nc  nl  numrowun  num_loads
2   2   128   256          32.0 Twr < P ==>btlnck Twr = 5.3   cacheusd(kbytes)= 2053.0    T16s(ms)= 175.7387
mr  nc  nl  numrowun  num_loads
2   2   128   512          16.0 Twr < P ==>btlnck Twr = 5.3   cacheusd(kbytes)= 4102.0    T16s(ms)= 175.5822
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   256   1          8192.0 Twr < P ==>btlnck Twr = 7.1   cacheusd(kbytes)= 12.0    T16s(ms)= 121.0951
mr  nc  nl  numrowun  num_loads
2   2   256   2          4096.0 Twr > P==>engh BUS BW Twr = 8.52   cacheusd(kbytes)= 20.0    T16s(ms)= 118.0294
mr  nc  nl  numrowun  num_loads
2   2   256   4          2048.0 Twr > P==>engh BUS BW Twr = 9.47   cacheusd(kbytes)= 36.0    T16s(ms)= 116.5928
mr  nc  nl  numrowun  num_loads
2   2   256   8          1024.0 Twr > P==>engh BUS BW Twr = 10.04   cacheusd(kbytes)= 68.0    T16s(ms)= 116.0680
mr  nc  nl  numrowun  num_loads
2   2   256   16          512.0 Twr > P==>engh BUS BW Twr = 10.34   cacheusd(kbytes)= 132.1    T16s(ms)= 116.1853
mr  nc  nl  numrowun  num_loads
2   2   256   32          256.0 Twr > P==>engh BUS BW Twr = 10.50   cacheusd(kbytes)= 260.1    T16s(ms)= 116.9905
mr  nc  nl  numrowun  num_loads
2   2   256   64          128.0 Twr > P==>engh BUS BW Twr = 10.59   cacheusd(kbytes)= 516.3    T16s(ms)= 118.8914
mr  nc  nl  numrowun  num_loads
2   2   256   128          64.0 Twr > P==>engh BUS BW Twr = 10.63   cacheusd(kbytes)= 1028.5    T16s(ms)= 122.8384
mr  nc  nl  numrowun  num_loads
2   2   256   256          32.0 Twr > P==>engh BUS BW Twr = 10.65   cacheusd(kbytes)= 2053.0    T16s(ms)= 130.8050
mr  nc  nl  numrowun  num_loads
2   2   256   512          16.0 Twr > P==>engh BUS BW Twr = 10.66   cacheusd(kbytes)= 4102.0    T16s(ms)= 129.5713
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   512   1          8192.0 Twr > P==>engh BUS BW Twr = 14.04   cacheusd(kbytes)= 12.0    T16s(ms)= 113.6336
mr  nc  nl  numrowun  num_loads
2   2   512   2          4096.0 Twr > P==>engh BUS BW Twr = 16.92   cacheusd(kbytes)= 20.0    T16s(ms)= 111.7960
mr  nc  nl  numrowun  num_loads
2   2   512   4          2048.0 Twr > P==>engh BUS BW Twr = 18.87   cacheusd(kbytes)= 36.0    T16s(ms)= 110.9240
mr  nc  nl  numrowun  num_loads
2   2   512   8          1024.0 Twr > P==>engh BUS BW Twr = 20.03   cacheusd(kbytes)= 68.0    T16s(ms)= 110.5816
mr  nc  nl  numrowun  num_loads
2   2   512   16          512.0 Twr > P==>engh BUS BW Twr = 20.67   cacheusd(kbytes)= 132.1    T16s(ms)= 110.5976
mr  nc  nl  numrowun  num_loads
2   2   512   32          256.0 Twr > P==>engh BUS BW Twr = 20.99   cacheusd(kbytes)= 260.1    T16s(ms)= 110.9829
mr  nc  nl  numrowun  num_loads
2   2   512   64          128.0 Twr > P==>engh BUS BW Twr = 21.16   cacheusd(kbytes)= 516.3    T16s(ms)= 111.9233
mr  nc  nl  numrowun  num_loads
2   2   512   128          64.0 Twr > P==>engh BUS BW Twr = 21.25   cacheusd(kbytes)= 1028.5    T16s(ms)= 113.8918
mr  nc  nl  numrowun  num_loads
```

```
2   2   512   256          32.0  Twr > P==>engh BUS BW Twr = 21.29   cacheusd(kbytes)= 2053.0     T16s(ms)= 117.8726
mr  nc  nl  numrowun  num_loads
2   2   512   512          16.0  Twr > P==>engh BUS BW Twr = 21.32   cacheusd(kbytes)= 4102.0     T16s(ms)= 117.2545
*********************************************************************************
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   32    1          8192.0  Twr < P ==>btlnck  Twr = 1.5    cacheusd(kbytes)= 16.0     T16s(ms)= 1260.9552
mr  nc  nl  numrowun  num_loads
3   3   32    2          4096.0  Twr < P ==>btlnck  Twr = 2.0    cacheusd(kbytes)= 24.0     T16s(ms)= 945.3584
mr  nc  nl  numrowun  num_loads
3   3   32    4          2048.0  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 40.0     T16s(ms)= 787.5600
mr  nc  nl  numrowun  num_loads
3   3   32    8          1024.0  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 72.1     T16s(ms)= 708.6608
mr  nc  nl  numrowun  num_loads
3   3   32    16          512.0  Twr < P ==>btlnck  Twr = 2.8    cacheusd(kbytes)= 136.1    T16s(ms)= 669.2112
mr  nc  nl  numrowun  num_loads
3   3   32    32          256.0  Twr < P ==>btlnck  Twr = 2.9    cacheusd(kbytes)= 264.3    T16s(ms)= 649.4864
mr  nc  nl  numrowun  num_loads
3   3   32    64          128.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 520.5    T16s(ms)= 639.6240
mr  nc  nl  numrowun  num_loads
3   3   32    128          64.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 1033.0    T16s(ms)= 634.6928
mr  nc  nl  numrowun  num_loads
3   3   32    256          32.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 2058.0    T16s(ms)= 632.2272
mr  nc  nl  numrowun  num_loads
3   3   32    512          16.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 4108.0    T16s(ms)= 630.9944
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   64    1          8192.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 16.0     T16s(ms)= 631.3879
mr  nc  nl  numrowun  num_loads
3   3   64    2          4096.0  Twr < P ==>btlnck  Twr = 4.0    cacheusd(kbytes)= 24.0     T16s(ms)= 473.2697
mr  nc  nl  numrowun  num_loads
3   3   64    4          2048.0  Twr < P ==>btlnck  Twr = 4.8    cacheusd(kbytes)= 40.0     T16s(ms)= 394.6524
mr  nc  nl  numrowun  num_loads
3   3   64    8          1024.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 72.1     T16s(ms)= 355.9202
mr  nc  nl  numrowun  num_loads
3   3   64    16          512.0  Twr < P ==>btlnck  Twr = 5.6    cacheusd(kbytes)= 136.1    T16s(ms)= 337.7070
mr  nc  nl  numrowun  num_loads
3   3   64    32          256.0  Twr < P ==>btlnck  Twr = 5.8    cacheusd(kbytes)= 264.3    T16s(ms)= 330.9062
mr  nc  nl  numrowun  num_loads
3   3   64    64          128.0  Twr < P ==>btlnck  Twr = 5.9    cacheusd(kbytes)= 520.5    T16s(ms)= 332.1174
mr  nc  nl  numrowun  num_loads
3   3   64    128          64.0  Twr < P ==>btlnck  Twr = 5.9    cacheusd(kbytes)= 1033.0    T16s(ms)= 341.9462
mr  nc  nl  numrowun  num_loads
3   3   64    256          32.0  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 2058.0    T16s(ms)= 365.3070
mr  nc  nl  numrowun  num_loads
3   3   64    512          16.0  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 4108.0    T16s(ms)= 364.6895
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   128   1          8192.0  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 16.0     T16s(ms)= 316.3351
mr  nc  nl  numrowun  num_loads
3   3   128   2          4096.0  Twr < P ==>btlnck  Twr = 8.0    cacheusd(kbytes)= 24.0     T16s(ms)= 266.6837
mr  nc  nl  numrowun  num_loads
3   3   128   4          2048.0  Twr > P==>engh BUS BW Twr = 9.59   cacheusd(kbytes)= 40.0     T16s(ms)= 261.5693
mr  nc  nl  numrowun  num_loads
3   3   128   8          1024.0  Twr > P==>engh BUS BW Twr = 10.66   cacheusd(kbytes)= 72.1     T16s(ms)= 259.4353
mr  nc  nl  numrowun  num_loads
3   3   128   16          512.0  Twr > P==>engh BUS BW Twr = 11.29   cacheusd(kbytes)= 136.1    T16s(ms)= 259.1178
mr  nc  nl  numrowun  num_loads
3   3   128   32          256.0  Twr > P==>engh BUS BW Twr = 11.63   cacheusd(kbytes)= 264.3    T16s(ms)= 260.4580
mr  nc  nl  numrowun  num_loads
3   3   128   64          128.0  Twr > P==>engh BUS BW Twr = 11.81   cacheusd(kbytes)= 520.5    T16s(ms)= 264.1260
mr  nc  nl  numrowun  num_loads
3   3   128   128          64.0  Twr > P==>engh BUS BW Twr = 11.90   cacheusd(kbytes)= 1033.0    T16s(ms)= 271.9558
mr  nc  nl  numrowun  num_loads
3   3   128   256          32.0  Twr > P==>engh BUS BW Twr = 11.95   cacheusd(kbytes)= 2058.0    T16s(ms)= 287.8623
mr  nc  nl  numrowun  num_loads
3   3   128   512          16.0  Twr > P==>engh BUS BW Twr = 11.97   cacheusd(kbytes)= 4108.0    T16s(ms)= 285.3924
```

```
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  256    1         8192.0  Twr > P==>engh BUS BW Twr = 11.96   cacheusd(kbytes)= 16.0     T16s(ms)= 257.1960
mr  nc  nl  numrowun  num_loads
3   3  256    2         4096.0  Twr > P==>engh BUS BW Twr = 15.95   cacheusd(kbytes)= 24.0     T16s(ms)= 251.6392
mr  nc  nl  numrowun  num_loads
3   3  256    4         2048.0  Twr > P==>engh BUS BW Twr = 19.16   cacheusd(kbytes)= 40.0     T16s(ms)= 248.9544
mr  nc  nl  numrowun  num_loads
3   3  256    8         1024.0  Twr > P==>engh BUS BW Twr = 21.29   cacheusd(kbytes)= 72.1     T16s(ms)= 247.8093
mr  nc  nl  numrowun  num_loads
3   3  256   16          512.0  Twr > P==>engh BUS BW Twr = 22.56   cacheusd(kbytes)= 136.1    T16s(ms)= 247.6065
mr  nc  nl  numrowun  num_loads
3   3  256   32          256.0  Twr > P==>engh BUS BW Twr = 23.25   cacheusd(kbytes)= 264.3    T16s(ms)= 248.2546
mr  nc  nl  numrowun  num_loads
3   3  256   64          128.0  Twr > P==>engh BUS BW Twr = 23.61   cacheusd(kbytes)= 520.5    T16s(ms)= 250.0776
mr  nc  nl  numrowun  num_loads
3   3  256  128           64.0  Twr > P==>engh BUS BW Twr = 23.80   cacheusd(kbytes)= 1033.0   T16s(ms)= 253.9870
mr  nc  nl  numrowun  num_loads
3   3  256  256           32.0  Twr > P==>engh BUS BW Twr = 23.89   cacheusd(kbytes)= 2058.0   T16s(ms)= 261.9375
mr  nc  nl  numrowun  num_loads
3   3  256  512           16.0  Twr > P==>engh BUS BW Twr = 23.94   cacheusd(kbytes)= 4108.0   T16s(ms)= 260.7012
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  512    1         8192.0  Twr > P==>engh BUS BW Twr = 23.70   cacheusd(kbytes)= 16.0     T16s(ms)= 247.3056
mr  nc  nl  numrowun  num_loads
3   3  512    2         4096.0  Twr > P==>engh BUS BW Twr = 31.71   cacheusd(kbytes)= 24.0     T16s(ms)= 244.1752
mr  nc  nl  numrowun  num_loads
3   3  512    4         2048.0  Twr > P==>engh BUS BW Twr = 38.18   cacheusd(kbytes)= 40.0     T16s(ms)= 242.6568
mr  nc  nl  numrowun  num_loads
3   3  512    8         1024.0  Twr > P==>engh BUS BW Twr = 42.54   cacheusd(kbytes)= 72.1     T16s(ms)= 241.9912
mr  nc  nl  numrowun  num_loads
3   3  512   16          512.0  Twr > P==>engh BUS BW Twr = 45.07   cacheusd(kbytes)= 136.1    T16s(ms)= 241.8509
mr  nc  nl  numrowun  num_loads
3   3  512   32          256.0  Twr > P==>engh BUS BW Twr = 46.47   cacheusd(kbytes)= 264.3    T16s(ms)= 242.1529
mr  nc  nl  numrowun  num_loads
3   3  512   64          128.0  Twr > P==>engh BUS BW Twr = 47.21   cacheusd(kbytes)= 520.5    T16s(ms)= 243.0534
mr  nc  nl  numrowun  num_loads
3   3  512  128           64.0  Twr > P==>engh BUS BW Twr = 47.59   cacheusd(kbytes)= 1033.0   T16s(ms)= 245.0026
mr  nc  nl  numrowun  num_loads
3   3  512  256           32.0  Twr > P==>engh BUS BW Twr = 47.78   cacheusd(kbytes)= 2058.0   T16s(ms)= 248.9751
mr  nc  nl  numrowun  num_loads
3   3  512  512           16.0  Twr > P==>engh BUS BW Twr = 47.87   cacheusd(kbytes)= 4108.0   T16s(ms)= 248.3556
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   32    1         8192.0  Twr < P ==>btlnck  Twr = 3.4    cacheusd(kbytes)= 28.1     T16s(ms)= 2213.3541
mr  nc  nl  numrowun  num_loads
6   6   32    2         4096.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 36.1     T16s(ms)= 1424.1318
mr  nc  nl  numrowun  num_loads
6   6   32    4         2048.0  Twr < P ==>btlnck  Twr = 7.4    cacheusd(kbytes)= 52.2     T16s(ms)= 1077.2199
mr  nc  nl  numrowun  num_loads
6   6   32    8         1024.0  Twr > P==>engh BUS BW Twr = 9.12   cacheusd(kbytes)= 84.3     T16s(ms)= 1052.8962
mr  nc  nl  numrowun  num_loads
6   6   32   16          512.0  Twr > P==>engh BUS BW Twr = 10.35   cacheusd(kbytes)= 148.4    T16s(ms)= 1044.2706
mr  nc  nl  numrowun  num_loads
6   6   32   32          256.0  Twr > P==>engh BUS BW Twr = 11.10   cacheusd(kbytes)= 276.7    T16s(ms)= 1046.0374
mr  nc  nl  numrowun  num_loads
6   6   32   64          128.0  Twr > P==>engh BUS BW Twr = 11.52   cacheusd(kbytes)= 533.3    T16s(ms)= 1058.9286
mr  nc  nl  numrowun  num_loads
6   6   32  128           64.0  Twr > P==>engh BUS BW Twr = 11.74   cacheusd(kbytes)= 1046.6   T16s(ms)= 1089.3898
mr  nc  nl  numrowun  num_loads
6   6   32  256           32.0  Twr > P==>engh BUS BW Twr = 11.86   cacheusd(kbytes)= 2073.1   T16s(ms)= 1152.6516
mr  nc  nl  numrowun  num_loads
6   6   32  512           16.0  Twr > P==>engh BUS BW Twr = 11.91   cacheusd(kbytes)= 4126.1   T16s(ms)= 1142.7193
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   64    1         8192.0  Twr < P ==>btlnck  Twr = 6.8    cacheusd(kbytes)= 28.1     T16s(ms)= 1108.3173
```

```
mr  nc  nl  numrowun  num_loads
6   6   64    2        4096.0  Twr > P==>engh BUS BW Twr = 10.63   cacheusd(kbytes)= 36.1     T16s(ms)= 1034.8876
mr  nc  nl  numrowun  num_loads
6   6   64    4        2048.0  Twr > P==>engh BUS BW Twr = 14.72   cacheusd(kbytes)= 52.2     T16s(ms)= 1010.0090
mr  nc  nl  numrowun  num_loads
6   6   64    8        1024.0  Twr > P==>engh BUS BW Twr = 18.23   cacheusd(kbytes)= 84.3     T16s(ms)= 998.4016
mr  nc  nl  numrowun  num_loads
6   6   64   16         512.0  Twr > P==>engh BUS BW Twr = 20.70   cacheusd(kbytes)= 148.4    T16s(ms)= 994.0989
mr  nc  nl  numrowun  num_loads
6   6   64   32         256.0  Twr > P==>engh BUS BW Twr = 22.20   cacheusd(kbytes)= 276.7    T16s(ms)= 994.9495
mr  nc  nl  numrowun  num_loads
6   6   64   64         128.0  Twr > P==>engh BUS BW Twr = 23.04   cacheusd(kbytes)= 533.3    T16s(ms)= 1001.3787
mr  nc  nl  numrowun  num_loads
6   6   64  128          64.0  Twr > P==>engh BUS BW Twr = 23.48   cacheusd(kbytes)= 1046.6   T16s(ms)= 1016.6011
mr  nc  nl  numrowun  num_loads
6   6   64  256          32.0  Twr > P==>engh BUS BW Twr = 23.71   cacheusd(kbytes)= 2073.1   T16s(ms)= 1048.2279
mr  nc  nl  numrowun  num_loads
6   6   64  512          16.0  Twr > P==>engh BUS BW Twr = 23.83   cacheusd(kbytes)= 4126.1   T16s(ms)= 1043.2597
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6  128    1        8192.0  Twr > P==>engh BUS BW Twr = 13.66   cacheusd(kbytes)= 28.1     T16s(ms)= 1015.3526
mr  nc  nl  numrowun  num_loads
6   6  128    2        4096.0  Twr > P==>engh BUS BW Twr = 21.24   cacheusd(kbytes)= 36.1     T16s(ms)= 989.7754
mr  nc  nl  numrowun  num_loads
6   6  128    4        2048.0  Twr > P==>engh BUS BW Twr = 29.43   cacheusd(kbytes)= 52.2     T16s(ms)= 977.1551
mr  nc  nl  numrowun  num_loads
6   6  128    8        1024.0  Twr > P==>engh BUS BW Twr = 36.44   cacheusd(kbytes)= 84.3     T16s(ms)= 971.2300
mr  nc  nl  numrowun  num_loads
6   6  128   16         512.0  Twr > P==>engh BUS BW Twr = 41.39   cacheusd(kbytes)= 148.4    T16s(ms)= 969.0131
mr  nc  nl  numrowun  num_loads
6   6  128   32         256.0  Twr > P==>engh BUS BW Twr = 44.40   cacheusd(kbytes)= 276.7    T16s(ms)= 969.4056
mr  nc  nl  numrowun  num_loads
6   6  128   64         128.0  Twr > P==>engh BUS BW Twr = 46.08   cacheusd(kbytes)= 533.3    T16s(ms)= 972.6038
mr  nc  nl  numrowun  num_loads
6   6  128  128          64.0  Twr > P==>engh BUS BW Twr = 46.97   cacheusd(kbytes)= 1046.6   T16s(ms)= 980.2068
mr  nc  nl  numrowun  num_loads
6   6  128  256          32.0  Twr > P==>engh BUS BW Twr = 47.42   cacheusd(kbytes)= 2073.1   T16s(ms)= 996.0161
mr  nc  nl  numrowun  num_loads
6   6  128  512          16.0  Twr > P==>engh BUS BW Twr = 47.65   cacheusd(kbytes)= 4126.1   T16s(ms)= 993.5299
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6  256    1        8192.0  Twr > P==>engh BUS BW Twr = 27.28   cacheusd(kbytes)= 28.1     T16s(ms)= 980.5853
mr  nc  nl  numrowun  num_loads
6   6  256    2        4096.0  Twr > P==>engh BUS BW Twr = 42.39   cacheusd(kbytes)= 36.1     T16s(ms)= 967.3105
mr  nc  nl  numrowun  num_loads
6   6  256    4        2048.0  Twr > P==>engh BUS BW Twr = 58.81   cacheusd(kbytes)= 52.2     T16s(ms)= 960.7281
mr  nc  nl  numrowun  num_loads
6   6  256    8        1024.0  Twr > P==>engh BUS BW Twr = 72.82   cacheusd(kbytes)= 84.3     T16s(ms)= 957.6442
mr  nc  nl  numrowun  num_loads
6   6  256   16         512.0  Twr > P==>engh BUS BW Twr = 82.75   cacheusd(kbytes)= 148.4    T16s(ms)= 956.4675
mr  nc  nl  numrowun  num_loads
6   6  256   32         256.0  Twr > P==>engh BUS BW Twr = 88.79   cacheusd(kbytes)= 276.7    T16s(ms)= 956.6322
mr  nc  nl  numrowun  num_loads
6   6  256   64         128.0  Twr > P==>engh BUS BW Twr = 92.15   cacheusd(kbytes)= 533.3    T16s(ms)= 958.2155
mr  nc  nl  numrowun  num_loads
6   6  256  128          64.0  Twr > P==>engh BUS BW Twr = 93.93   cacheusd(kbytes)= 1046.6   T16s(ms)= 962.0091
mr  nc  nl  numrowun  num_loads
6   6  256  256          32.0  Twr > P==>engh BUS BW Twr = 94.84   cacheusd(kbytes)= 2073.1   T16s(ms)= 969.9098
mr  nc  nl  numrowun  num_loads
6   6  256  512          16.0  Twr > P==>engh BUS BW Twr = 95.31   cacheusd(kbytes)= 4126.1   T16s(ms)= 968.6647
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6  512    1        8192.0  Twr > P==>engh BUS BW Twr = 54.41   cacheusd(kbytes)= 28.1     T16s(ms)= 963.2016
mr  nc  nl  numrowun  num_loads
6   6  512    2        4096.0  Twr > P==>engh BUS BW Twr = 84.45   cacheusd(kbytes)= 36.1     T16s(ms)= 956.0781
mr  nc  nl  numrowun  num_loads
6   6  512    4        2048.0  Twr > P==>engh BUS BW Twr = 117.28  cacheusd(kbytes)= 52.2     T16s(ms)= 952.5245
```

```
mr  nc  nl  numrowun  num_loads
6   6   512   8          1024.0  Twr > P==>engh BUS BW Twr = 145.40   cacheusd(kbytes)= 84.3     T16s(ms)= 950.8513
mr  nc  nl  numrowun  num_loads
6   6   512   16         512.0  Twr > P==>engh BUS BW Twr = 165.34   cacheusd(kbytes)= 148.4     T16s(ms)= 950.1974
mr  nc  nl  numrowun  num_loads
6   6   512   32         256.0  Twr > P==>engh BUS BW Twr = 177.52   cacheusd(kbytes)= 276.7     T16s(ms)= 950.2455
mr  nc  nl  numrowun  num_loads
6   6   512   64         128.0  Twr > P==>engh BUS BW Twr = 184.27   cacheusd(kbytes)= 533.3     T16s(ms)= 951.0213
mr  nc  nl  numrowun  num_loads
6   6   512   128         64.0  Twr > P==>engh BUS BW Twr = 187.84   cacheusd(kbytes)= 1046.6     T16s(ms)= 952.9102
mr  nc  nl  numrowun  num_loads
6   6   512   256         32.0  Twr > P==>engh BUS BW Twr = 189.67   cacheusd(kbytes)= 2073.1     T16s(ms)= 956.8566
mr  nc  nl  numrowun  num_loads
6   6   512   512         16.0  Twr > P==>engh BUS BW Twr = 190.61   cacheusd(kbytes)= 4126.1     T16s(ms)= 956.2321
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   32    1          8192.0  Twr < P ==>btlnck  Twr = 5.4    cacheusd(kbytes)= 40.3     T16s(ms)= 3172.2811
mr  nc  nl  numrowun  num_loads
9   9   32    2          4096.0  Twr > P==>engh BUS BW Twr = 8.95   cacheusd(kbytes)= 48.3     T16s(ms)= 2367.0121
mr  nc  nl  numrowun  num_loads
9   9   32    4          2048.0  Twr > P==>engh BUS BW Twr = 13.42   cacheusd(kbytes)= 64.4     T16s(ms)= 2286.3068
mr  nc  nl  numrowun  num_loads
9   9   32    8          1024.0  Twr > P==>engh BUS BW Twr = 17.91   cacheusd(kbytes)= 96.5     T16s(ms)= 2248.2852
mr  nc  nl  numrowun  num_loads
9   9   32    16          512.0  Twr > P==>engh BUS BW Twr = 21.50   cacheusd(kbytes)= 160.8     T16s(ms)= 2232.2804
mr  nc  nl  numrowun  num_loads
9   9   32    32          256.0  Twr > P==>engh BUS BW Twr = 23.90   cacheusd(kbytes)= 289.2     T16s(ms)= 2230.2900
mr  nc  nl  numrowun  num_loads
9   9   32    64          128.0  Twr > P==>engh BUS BW Twr = 25.31   cacheusd(kbytes)= 546.2     T16s(ms)= 2241.3188
mr  nc  nl  numrowun  num_loads
9   9   32    128          64.0  Twr > P==>engh BUS BW Twr = 26.08   cacheusd(kbytes)= 1060.2     T16s(ms)= 2270.8812
mr  nc  nl  numrowun  num_loads
9   9   32    256          32.0  Twr > P==>engh BUS BW Twr = 26.48   cacheusd(kbytes)= 2088.2     T16s(ms)= 2333.7584
mr  nc  nl  numrowun  num_loads
9   9   32    512          16.0  Twr > P==>engh BUS BW Twr = 26.69   cacheusd(kbytes)= 4144.2     T16s(ms)= 2323.7634
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   64    1          8192.0  Twr > P==>engh BUS BW Twr = 10.73   cacheusd(kbytes)= 40.3     T16s(ms)= 2326.3248
mr  nc  nl  numrowun  num_loads
9   9   64    2          4096.0  Twr > P==>engh BUS BW Twr = 17.89   cacheusd(kbytes)= 48.3     T16s(ms)= 2245.1934
mr  nc  nl  numrowun  num_loads
9   9   64    4          2048.0  Twr > P==>engh BUS BW Twr = 26.84   cacheusd(kbytes)= 64.4     T16s(ms)= 2205.2720
mr  nc  nl  numrowun  num_loads
9   9   64    8          1024.0  Twr > P==>engh BUS BW Twr = 35.81   cacheusd(kbytes)= 96.5     T16s(ms)= 2186.0628
mr  nc  nl  numrowun  num_loads
9   9   64    16          512.0  Twr > P==>engh BUS BW Twr = 43.00   cacheusd(kbytes)= 160.8     T16s(ms)= 2177.9612
mr  nc  nl  numrowun  num_loads
9   9   64    32          256.0  Twr > P==>engh BUS BW Twr = 47.79   cacheusd(kbytes)= 289.2     T16s(ms)= 2176.9164
mr  nc  nl  numrowun  num_loads
9   9   64    64          128.0  Twr > P==>engh BUS BW Twr = 50.62   cacheusd(kbytes)= 546.2     T16s(ms)= 2182.4060
mr  nc  nl  numrowun  num_loads
9   9   64    128          64.0  Twr > P==>engh BUS BW Twr = 52.16   cacheusd(kbytes)= 1060.2     T16s(ms)= 2197.1748
mr  nc  nl  numrowun  num_loads
9   9   64    256          32.0  Twr > P==>engh BUS BW Twr = 52.96   cacheusd(kbytes)= 2088.2     T16s(ms)= 2228.6072
mr  nc  nl  numrowun  num_loads
9   9   64    512          16.0  Twr > P==>engh BUS BW Twr = 53.37   cacheusd(kbytes)= 4144.2     T16s(ms)= 2223.6066
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   128   1          8192.0  Twr > P==>engh BUS BW Twr = 21.46   cacheusd(kbytes)= 40.3     T16s(ms)= 2226.2028
mr  nc  nl  numrowun  num_loads
9   9   128   2          4096.0  Twr > P==>engh BUS BW Twr = 35.76   cacheusd(kbytes)= 48.3     T16s(ms)= 2185.1121
mr  nc  nl  numrowun  num_loads
9   9   128   4          2048.0  Twr > P==>engh BUS BW Twr = 53.67   cacheusd(kbytes)= 64.4     T16s(ms)= 2164.7546
mr  nc  nl  numrowun  num_loads
9   9   128   8          1024.0  Twr > P==>engh BUS BW Twr = 71.61   cacheusd(kbytes)= 96.5     T16s(ms)= 2154.9516
mr  nc  nl  numrowun  num_loads
```

```
9   9   128   16          512.0  Twr > P==>engh BUS BW Twr = 85.99   cacheusd(kbytes)= 160.8     T16s(ms)= 2150.8016
mr  nc  nl  numrowun  num_loads
9   9   128   32          256.0  Twr > P==>engh BUS BW Twr = 95.58   cacheusd(kbytes)= 289.2     T16s(ms)= 2150.2296
mr  nc  nl  numrowun  num_loads
9   9   128   64          128.0  Twr > P==>engh BUS BW Twr = 101.23  cacheusd(kbytes)= 546.2     T16s(ms)= 2152.9496
mr  nc  nl  numrowun  num_loads
9   9   128   128          64.0  Twr > P==>engh BUS BW Twr = 104.31  cacheusd(kbytes)= 1060.2    T16s(ms)= 2160.3216
mr  nc  nl  numrowun  num_loads
9   9   128   256          32.0  Twr > P==>engh BUS BW Twr = 105.92  cacheusd(kbytes)= 2088.2    T16s(ms)= 2176.0316
mr  nc  nl  numrowun  num_loads
9   9   128   512          16.0  Twr > P==>engh BUS BW Twr = 106.75  cacheusd(kbytes)= 4144.2    T16s(ms)= 2173.5282
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256   1          8192.0  Twr > P==>engh BUS BW Twr = 42.84   cacheusd(kbytes)= 40.3      T16s(ms)= 2176.4490
mr  nc  nl  numrowun  num_loads
9   9   256   2          4096.0  Twr > P==>engh BUS BW Twr = 71.47   cacheusd(kbytes)= 48.3      T16s(ms)= 2155.0714
mr  nc  nl  numrowun  num_loads
9   9   256   4          2048.0  Twr > P==>engh BUS BW Twr = 107.27  cacheusd(kbytes)= 64.4      T16s(ms)= 2144.4959
mr  nc  nl  numrowun  num_loads
9   9   256   8          1024.0  Twr > P==>engh BUS BW Twr = 143.16  cacheusd(kbytes)= 96.5      T16s(ms)= 2139.3960
mr  nc  nl  numrowun  num_loads
9   9   256   16          512.0  Twr > P==>engh BUS BW Twr = 171.93  cacheusd(kbytes)= 160.8     T16s(ms)= 2137.2218
mr  nc  nl  numrowun  num_loads
9   9   256   32          256.0  Twr > P==>engh BUS BW Twr = 191.13  cacheusd(kbytes)= 289.2     T16s(ms)= 2136.8862
mr  nc  nl  numrowun  num_loads
9   9   256   64          128.0  Twr > P==>engh BUS BW Twr = 202.44  cacheusd(kbytes)= 546.2     T16s(ms)= 2138.2214
mr  nc  nl  numrowun  num_loads
9   9   256   128          64.0  Twr > P==>engh BUS BW Twr = 208.61  cacheusd(kbytes)= 1060.2    T16s(ms)= 2141.8950
mr  nc  nl  numrowun  num_loads
9   9   256   256          32.0  Twr > P==>engh BUS BW Twr = 211.84  cacheusd(kbytes)= 2088.2    T16s(ms)= 2149.7438
mr  nc  nl  numrowun  num_loads
9   9   256   512          16.0  Twr > P==>engh BUS BW Twr = 213.49  cacheusd(kbytes)= 4144.2    T16s(ms)= 2148.4890
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   512   1          8192.0  Twr > P==>engh BUS BW Twr = 85.37   cacheusd(kbytes)= 40.3      T16s(ms)= 2151.5721
mr  nc  nl  numrowun  num_loads
9   9   512   2          4096.0  Twr > P==>engh BUS BW Twr = 142.50  cacheusd(kbytes)= 48.3      T16s(ms)= 2140.0705
mr  nc  nl  numrowun  num_loads
9   9   512   4          2048.0  Twr > P==>engh BUS BW Twr = 214.27  cacheusd(kbytes)= 64.4      T16s(ms)= 2134.3665
mr  nc  nl  numrowun  num_loads
9   9   512   8          1024.0  Twr > P==>engh BUS BW Twr = 286.08  cacheusd(kbytes)= 96.5      T16s(ms)= 2131.6182
mr  nc  nl  numrowun  num_loads
9   9   512   16          512.0  Twr > P==>engh BUS BW Twr = 343.67  cacheusd(kbytes)= 160.8     T16s(ms)= 2130.4319
mr  nc  nl  numrowun  num_loads
9   9   512   32          256.0  Twr > P==>engh BUS BW Twr = 382.15  cacheusd(kbytes)= 289.2     T16s(ms)= 2130.2145
mr  nc  nl  numrowun  num_loads
9   9   512   64          128.0  Twr > P==>engh BUS BW Twr = 404.81  cacheusd(kbytes)= 546.2     T16s(ms)= 2130.8573
mr  nc  nl  numrowun  num_loads
9   9   512   128          64.0  Twr > P==>engh BUS BW Twr = 417.18  cacheusd(kbytes)= 1060.2    T16s(ms)= 2132.6817
mr  nc  nl  numrowun  num_loads
9   9   512   256          32.0  Twr > P==>engh BUS BW Twr = 423.66  cacheusd(kbytes)= 2088.2    T16s(ms)= 2136.5999
mr  nc  nl  numrowun  num_loads
9   9   512   512          16.0  Twr > P==>engh BUS BW Twr = 426.97  cacheusd(kbytes)= 4144.2    T16s(ms)= 2135.9694
*********************************************************************************
 *********************************************************************************
```

P=16

```
mr  nc  nl  numrowun
2   2   32    1  Twr < P ==>btlnck  Twr = 0.9    cacheusd(kbytes)= 12.0     T32s(ms)= 945.1552
mr  nc  nl  numrowun
2   2   32    2  Twr < P ==>btlnck  Twr = 1.1    cacheusd(kbytes)= 20.0     T32s(ms)= 787.4592
mr  nc  nl  numrowun
2   2   32    4  Twr < P ==>btlnck  Twr = 1.2    cacheusd(kbytes)= 36.0     T32s(ms)= 708.4576
mr  nc  nl  numrowun
2   2   32    8  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 68.0     T32s(ms)= 668.9568
mr  nc  nl  numrowun
2   2   32    16  Twr < P ==>btlnck  Twr = 1.3   cacheusd(kbytes)= 132.1    T32s(ms)= 649.2064
```

```
mr  nc  nl  numrowun
2   2   32   32  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 260.1     T32s(ms)= 639.3312
mr  nc  nl  numrowun
2   2   32   64  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 516.3     T32s(ms)= 634.3936
mr  nc  nl  numrowun
2   2   32   128  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 1028.5     T32s(ms)= 631.9248
mr  nc  nl  numrowun
2   2   32   256  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 2053.0     T32s(ms)= 630.6904
*********************************************************************************
 mr  nc  nl  numrowun
2   2   64    1  Twr < P ==>btlnck  Twr = 1.8    cacheusd(kbytes)= 12.0     T32s(ms)= 473.2960
mr  nc  nl  numrowun
2   2   64    2  Twr < P ==>btlnck  Twr = 2.1    cacheusd(kbytes)= 20.0     T32s(ms)= 393.9360
mr  nc  nl  numrowun
2   2   64    4  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 36.0     T32s(ms)= 354.4096
mr  nc  nl  numrowun
2   2   64    8  Twr < P ==>btlnck  Twr = 2.5    cacheusd(kbytes)= 68.0     T32s(ms)= 334.5696
mr  nc  nl  numrowun
2   2   64   16  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 132.1     T32s(ms)= 324.6496
mr  nc  nl  numrowun
2   2   64   32  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 260.1     T32s(ms)= 319.6896
mr  nc  nl  numrowun
2   2   64   64  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 516.3     T32s(ms)= 317.2096
mr  nc  nl  numrowun
2   2   64   128  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 1028.5     T32s(ms)= 315.9696
mr  nc  nl  numrowun
2   2   64   256  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 2053.0     T32s(ms)= 315.3496
*********************************************************************************
 mr  nc  nl  numrowun
2   2   128    1  Twr < P ==>btlnck  Twr = 3.6    cacheusd(kbytes)= 12.0     T32s(ms)= 237.3664
mr  nc  nl  numrowun
2   2   128    2  Twr < P ==>btlnck  Twr = 4.3    cacheusd(kbytes)= 20.0     T32s(ms)= 197.3280
mr  nc  nl  numrowun
2   2   128    4  Twr < P ==>btlnck  Twr = 4.7    cacheusd(kbytes)= 36.0     T32s(ms)= 177.3088
mr  nc  nl  numrowun
2   2   128    8  Twr < P ==>btlnck  Twr = 5.0    cacheusd(kbytes)= 68.0     T32s(ms)= 167.3760
mr  nc  nl  numrowun
2   2   128   16  Twr < P ==>btlnck  Twr = 5.2    cacheusd(kbytes)= 132.1     T32s(ms)= 162.3712
mr  nc  nl  numrowun
2   2   128   32  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 260.1     T32s(ms)= 159.8688
mr  nc  nl  numrowun
2   2   128   64  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 516.3     T32s(ms)= 158.6176
mr  nc  nl  numrowun
2   2   128   128  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 1028.5     T32s(ms)= 157.9920
mr  nc  nl  numrowun
2   2   128   256  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 2053.0     T32s(ms)= 157.6792
*********************************************************************************
 mr  nc  nl  numrowun
2   2   256    1  Twr < P ==>btlnck  Twr = 7.1    cacheusd(kbytes)= 12.0     T32s(ms)= 119.4613
mr  nc  nl  numrowun
2   2   256    2  Twr < P ==>btlnck  Twr = 8.5    cacheusd(kbytes)= 20.0     T32s(ms)= 99.1446
mr  nc  nl  numrowun
2   2   256    4  Twr < P ==>btlnck  Twr = 9.5    cacheusd(kbytes)= 36.0     T32s(ms)= 89.0778
mr  nc  nl  numrowun
2   2   256    8  Twr < P ==>btlnck  Twr = 10.0    cacheusd(kbytes)= 68.0     T32s(ms)= 84.2272
mr  nc  nl  numrowun
2   2   256   16  Twr < P ==>btlnck  Twr = 10.3    cacheusd(kbytes)= 132.1     T32s(ms)= 82.2060
mr  nc  nl  numrowun
2   2   256   32  Twr < P ==>btlnck  Twr = 10.5    cacheusd(kbytes)= 260.1     T32s(ms)= 81.9076
mr  nc  nl  numrowun
2   2   256   64  Twr < P ==>btlnck  Twr = 10.6    cacheusd(kbytes)= 516.3     T32s(ms)= 83.2212
mr  nc  nl  numrowun
2   2   256   128  Twr < P ==>btlnck  Twr = 10.6    cacheusd(kbytes)= 1028.5     T32s(ms)= 86.8036
mr  nc  nl  numrowun
2   2   256   256  Twr < P ==>btlnck  Twr = 10.6    cacheusd(kbytes)= 2053.0     T32s(ms)= 86.6450
*********************************************************************************
 mr  nc  nl  numrowun
```

```
2   2   512    1  Twr < P ==>btlnck  Twr = 14.0    cacheusd(kbytes)= 12.0     T32s(ms)= 60.5509
mr  nc  nl  numrowun
2   2   512    2  Twr > P==>engh BUS BW Twr = 16.92   cacheusd(kbytes)= 20.0     T32s(ms)= 56.0569
mr  nc  nl  numrowun
2   2   512    4  Twr > P==>engh BUS BW Twr = 18.87   cacheusd(kbytes)= 36.0     T32s(ms)= 55.7063
mr  nc  nl  numrowun
2   2   512    8  Twr > P==>engh BUS BW Twr = 20.03   cacheusd(kbytes)= 68.0     T32s(ms)= 55.7510
mr  nc  nl  numrowun
2   2   512    16  Twr > P==>engh BUS BW Twr = 20.67   cacheusd(kbytes)= 132.1     T32s(ms)= 56.1911
mr  nc  nl  numrowun
2   2   512    32  Twr > P==>engh BUS BW Twr = 20.99   cacheusd(kbytes)= 260.1     T32s(ms)= 57.2485
mr  nc  nl  numrowun
2   2   512    64  Twr > P==>engh BUS BW Twr = 21.16   cacheusd(kbytes)= 516.3     T32s(ms)= 59.4475
mr  nc  nl  numrowun
2   2   512    128  Twr > P==>engh BUS BW Twr = 21.25   cacheusd(kbytes)= 1028.5     T32s(ms)= 63.8895
mr  nc  nl  numrowun
2   2   512    256  Twr > P==>engh BUS BW Twr = 21.29   cacheusd(kbytes)= 2053.0     T32s(ms)= 63.5792
*****************************************************************************
*****************************************************************************
 mr  nc  nl  numrowun
3   3   32    1  Twr < P ==>btlnck  Twr = 1.5    cacheusd(kbytes)= 16.0     T32s(ms)= 1260.9568
mr  nc  nl  numrowun
3   3   32    2  Twr < P ==>btlnck  Twr = 2.0    cacheusd(kbytes)= 24.0     T32s(ms)= 945.3600
mr  nc  nl  numrowun
3   3   32    4  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 40.0     T32s(ms)= 787.5616
mr  nc  nl  numrowun
3   3   32    8  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 72.1     T32s(ms)= 708.6624
mr  nc  nl  numrowun
3   3   32    16  Twr < P ==>btlnck  Twr = 2.8    cacheusd(kbytes)= 136.1     T32s(ms)= 669.2128
mr  nc  nl  numrowun
3   3   32    32  Twr < P ==>btlnck  Twr = 2.9    cacheusd(kbytes)= 264.3     T32s(ms)= 649.4880
mr  nc  nl  numrowun
3   3   32    64  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 520.5     T32s(ms)= 639.6256
mr  nc  nl  numrowun
3   3   32    128  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 1033.0     T32s(ms)= 634.6944
mr  nc  nl  numrowun
3   3   32    256  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 2058.0     T32s(ms)= 632.2288
*****************************************************************************
 mr  nc  nl  numrowun
3   3   64    1  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 16.0     T32s(ms)= 631.1968
mr  nc  nl  numrowun
3   3   64    2  Twr < P ==>btlnck  Twr = 4.0    cacheusd(kbytes)= 24.0     T32s(ms)= 472.8864
mr  nc  nl  numrowun
3   3   64    4  Twr < P ==>btlnck  Twr = 4.8    cacheusd(kbytes)= 40.0     T32s(ms)= 393.8848
mr  nc  nl  numrowun
3   3   64    8  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 72.1     T32s(ms)= 354.3840
mr  nc  nl  numrowun
3   3   64    16  Twr < P ==>btlnck  Twr = 5.6    cacheusd(kbytes)= 136.1     T32s(ms)= 334.6336
mr  nc  nl  numrowun
3   3   64    32  Twr < P ==>btlnck  Twr = 5.8    cacheusd(kbytes)= 264.3     T32s(ms)= 324.7584
mr  nc  nl  numrowun
3   3   64    64  Twr < P ==>btlnck  Twr = 5.9    cacheusd(kbytes)= 520.5     T32s(ms)= 319.8208
mr  nc  nl  numrowun
3   3   64    128  Twr < P ==>btlnck  Twr = 5.9    cacheusd(kbytes)= 1033.0     T32s(ms)= 317.3520
mr  nc  nl  numrowun
3   3   64    256  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 2058.0     T32s(ms)= 316.1176
*****************************************************************************
 mr  nc  nl  numrowun
3   3   128    1  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 16.0     T32s(ms)= 316.1816
mr  nc  nl  numrowun
3   3   128    2  Twr < P ==>btlnck  Twr = 8.0    cacheusd(kbytes)= 24.0     T32s(ms)= 236.9945
mr  nc  nl  numrowun
3   3   128    4  Twr < P ==>btlnck  Twr = 9.6    cacheusd(kbytes)= 40.0     T32s(ms)= 197.8140
mr  nc  nl  numrowun
3   3   128    8  Twr < P ==>btlnck  Twr = 10.7    cacheusd(kbytes)= 72.1     T32s(ms)= 178.6658
mr  nc  nl  numrowun
3   3   128    16  Twr < P ==>btlnck  Twr = 11.3    cacheusd(kbytes)= 136.1     T32s(ms)= 170.1294
```

```
mr  nc  nl  numrowun
3   3   128   32  Twr < P ==>btlnck  Twr = 11.6    cacheusd(kbytes)= 264.3     T32s(ms)= 167.9366
mr  nc  nl  numrowun
3   3   128   64  Twr < P ==>btlnck  Twr = 11.8    cacheusd(kbytes)= 520.5     T32s(ms)= 170.9910
mr  nc  nl  numrowun
3   3   128   128 Twr < P ==>btlnck  Twr = 11.9    cacheusd(kbytes)= 1033.0    T32s(ms)= 180.8198
mr  nc  nl  numrowun
3   3   128   256 Twr < P ==>btlnck  Twr = 11.9    cacheusd(kbytes)= 2058.0    T32s(ms)= 180.2003
*******************************************************************************

 mr  nc  nl  numrowun
3   3   256    1  Twr < P ==>btlnck  Twr = 12.0    cacheusd(kbytes)= 16.0      T32s(ms)= 159.0392
mr  nc  nl  numrowun
3   3   256    2  Twr < P ==>btlnck  Twr = 15.9    cacheusd(kbytes)= 24.0      T32s(ms)= 126.2815
mr  nc  nl  numrowun
3   3   256    4  Twr > P==>engh BUS BW Twr = 19.16   cacheusd(kbytes)= 40.0     T32s(ms)= 125.0167
mr  nc  nl  numrowun
3   3   256    8  Twr > P==>engh BUS BW Twr = 21.29   cacheusd(kbytes)= 72.1     T32s(ms)= 124.8769
mr  nc  nl  numrowun
3   3   256    16 Twr > P==>engh BUS BW Twr = 22.56   cacheusd(kbytes)= 136.1    T32s(ms)= 125.6404
mr  nc  nl  numrowun
3   3   256    32 Twr > P==>engh BUS BW Twr = 23.25   cacheusd(kbytes)= 264.3    T32s(ms)= 127.6941
mr  nc  nl  numrowun
3   3   256    64 Twr > P==>engh BUS BW Twr = 23.61   cacheusd(kbytes)= 520.5    T32s(ms)= 132.0649
mr  nc  nl  numrowun
3   3   256    128 Twr > P==>engh BUS BW Twr = 23.80   cacheusd(kbytes)= 1033.0    T32s(ms)= 140.9382
mr  nc  nl  numrowun
3   3   256    256 Twr > P==>engh BUS BW Twr = 23.89   cacheusd(kbytes)= 2058.0    T32s(ms)= 140.3186
*******************************************************************************

 mr  nc  nl  numrowun
3   3   512    1  Twr > P==>engh BUS BW Twr = 23.70   cacheusd(kbytes)= 16.0      T32s(ms)= 123.7615
mr  nc  nl  numrowun
3   3   512    2  Twr > P==>engh BUS BW Twr = 31.71   cacheusd(kbytes)= 24.0      T32s(ms)= 122.2503
mr  nc  nl  numrowun
3   3   512    4  Twr > P==>engh BUS BW Twr = 38.18   cacheusd(kbytes)= 40.0      T32s(ms)= 121.5991
mr  nc  nl  numrowun
3   3   512    8  Twr > P==>engh BUS BW Twr = 42.54   cacheusd(kbytes)= 72.1      T32s(ms)= 121.4823
mr  nc  nl  numrowun
3   3   512    16 Twr > P==>engh BUS BW Twr = 45.07   cacheusd(kbytes)= 136.1     T32s(ms)= 121.8449
mr  nc  nl  numrowun
3   3   512    32 Twr > P==>engh BUS BW Twr = 46.47   cacheusd(kbytes)= 264.3     T32s(ms)= 122.8608
mr  nc  nl  numrowun
3   3   512    64 Twr > P==>engh BUS BW Twr = 47.21   cacheusd(kbytes)= 520.5     T32s(ms)= 125.0406
mr  nc  nl  numrowun
3   3   512    128 Twr > P==>engh BUS BW Twr = 47.59   cacheusd(kbytes)= 1033.0    T32s(ms)= 129.4745
mr  nc  nl  numrowun
3   3   512    256 Twr > P==>engh BUS BW Twr = 47.78   cacheusd(kbytes)= 2058.0    T32s(ms)= 129.1634
*******************************************************************************
*******************************************************************************

 mr  nc  nl  numrowun
6   6   32     1  Twr < P ==>btlnck  Twr = 3.4    cacheusd(kbytes)= 28.1      T32s(ms)= 2212.7397
mr  nc  nl  numrowun
6   6   32     2  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 36.1      T32s(ms)= 1422.9031
mr  nc  nl  numrowun
6   6   32     4  Twr < P ==>btlnck  Twr = 7.4    cacheusd(kbytes)= 52.2      T32s(ms)= 1028.8682
mr  nc  nl  numrowun
6   6   32     8  Twr < P ==>btlnck  Twr = 9.1    cacheusd(kbytes)= 84.3      T32s(ms)= 833.9248
mr  nc  nl  numrowun
6   6   32     16 Twr < P ==>btlnck  Twr = 10.4    cacheusd(kbytes)= 148.4     T32s(ms)= 740.6012
mr  nc  nl  numrowun
6   6   32     32 Twr < P ==>btlnck  Twr = 11.1    cacheusd(kbytes)= 276.7     T32s(ms)= 702.2355
mr  nc  nl  numrowun
6   6   32     64 Twr < P ==>btlnck  Twr = 11.5    cacheusd(kbytes)= 533.3     T32s(ms)= 699.6452
mr  nc  nl  numrowun
6   6   32     128 Twr < P ==>btlnck  Twr = 11.7    cacheusd(kbytes)= 1046.6    T32s(ms)= 731.5348
mr  nc  nl  numrowun
6   6   32     256 Twr < P ==>btlnck  Twr = 11.9    cacheusd(kbytes)= 2073.1    T32s(ms)= 725.3562
*******************************************************************************
```

```
 mr  nc  nl  numrowun
6    6   64     1  Twr < P ==>btlnck  Twr = 6.8     cacheusd(kbytes)= 28.1      T32s(ms)= 1108.0101
mr  nc  nl  numrowun
6    6   64     2  Twr < P ==>btlnck  Twr = 10.6     cacheusd(kbytes)= 36.1      T32s(ms)= 713.5015
mr  nc  nl  numrowun
6    6   64     4  Twr < P ==>btlnck  Twr = 14.7     cacheusd(kbytes)= 52.2      T32s(ms)= 518.3017
mr  nc  nl  numrowun
6    6   64     8  Twr > P==>engh BUS BW Twr = 18.23    cacheusd(kbytes)= 84.3      T32s(ms)= 504.1328
mr  nc  nl  numrowun
6    6   64    16  Twr > P==>engh BUS BW Twr = 20.70    cacheusd(kbytes)= 148.4      T32s(ms)= 505.0408
mr  nc  nl  numrowun
6    6   64    32  Twr > P==>engh BUS BW Twr = 22.20    cacheusd(kbytes)= 276.7      T32s(ms)= 512.3945
mr  nc  nl  numrowun
6    6   64    64  Twr > P==>engh BUS BW Twr = 23.04    cacheusd(kbytes)= 533.3      T32s(ms)= 529.4661
mr  nc  nl  numrowun
6    6   64   128  Twr > P==>engh BUS BW Twr = 23.48    cacheusd(kbytes)= 1046.6      T32s(ms)= 564.7914
mr  nc  nl  numrowun
6    6   64   256  Twr > P==>engh BUS BW Twr = 23.71    cacheusd(kbytes)= 2073.1      T32s(ms)= 562.3048
*********************************************************************************
 mr  nc  nl  numrowun
6    6  128     1  Twr < P ==>btlnck  Twr = 13.7     cacheusd(kbytes)= 28.1      T32s(ms)= 555.3381
mr  nc  nl  numrowun
6    6  128     2  Twr > P==>engh BUS BW Twr = 21.24    cacheusd(kbytes)= 36.1      T32s(ms)= 496.1056
mr  nc  nl  numrowun
6    6  128     4  Twr > P==>engh BUS BW Twr = 29.43    cacheusd(kbytes)= 52.2      T32s(ms)= 489.9755
mr  nc  nl  numrowun
6    6  128     8  Twr > P==>engh BUS BW Twr = 36.44    cacheusd(kbytes)= 84.3      T32s(ms)= 487.8795
mr  nc  nl  numrowun
6    6  128    16  Twr > P==>engh BUS BW Twr = 41.39    cacheusd(kbytes)= 148.4      T32s(ms)= 488.5031
mr  nc  nl  numrowun
6    6  128    32  Twr > P==>engh BUS BW Twr = 44.40    cacheusd(kbytes)= 276.7      T32s(ms)= 492.1636
mr  nc  nl  numrowun
6    6  128    64  Twr > P==>engh BUS BW Twr = 46.08    cacheusd(kbytes)= 533.3      T32s(ms)= 500.6912
mr  nc  nl  numrowun
6    6  128   128  Twr > P==>engh BUS BW Twr = 46.97    cacheusd(kbytes)= 1046.6      T32s(ms)= 518.3497
mr  nc  nl  numrowun
6    6  128   256  Twr > P==>engh BUS BW Twr = 47.42    cacheusd(kbytes)= 2073.1      T32s(ms)= 517.1044
*********************************************************************************
 mr  nc  nl  numrowun
6    6  256     1  Twr > P==>engh BUS BW Twr = 27.28    cacheusd(kbytes)= 28.1      T32s(ms)= 490.6673
mr  nc  nl  numrowun
6    6  256     2  Twr > P==>engh BUS BW Twr = 42.39    cacheusd(kbytes)= 36.1      T32s(ms)= 484.1388
mr  nc  nl  numrowun
6    6  256     4  Twr > P==>engh BUS BW Twr = 58.81    cacheusd(kbytes)= 52.2      T32s(ms)= 481.0636
mr  nc  nl  numrowun
6    6  256     8  Twr > P==>engh BUS BW Twr = 72.82    cacheusd(kbytes)= 84.3      T32s(ms)= 479.9553
mr  nc  nl  numrowun
6    6  256    16  Twr > P==>engh BUS BW Twr = 82.75    cacheusd(kbytes)= 148.4      T32s(ms)= 480.2326
mr  nc  nl  numrowun
6    6  256    32  Twr > P==>engh BUS BW Twr = 88.79    cacheusd(kbytes)= 276.7      T32s(ms)= 482.0470
mr  nc  nl  numrowun
6    6  256    64  Twr > P==>engh BUS BW Twr = 92.15    cacheusd(kbytes)= 533.3      T32s(ms)= 486.3029
mr  nc  nl  numrowun
6    6  256   128  Twr > P==>engh BUS BW Twr = 93.93    cacheusd(kbytes)= 1046.6      T32s(ms)= 495.1282
mr  nc  nl  numrowun
6    6  256   256  Twr > P==>engh BUS BW Twr = 94.84    cacheusd(kbytes)= 2073.1      T32s(ms)= 494.5036
*********************************************************************************
 mr  nc  nl  numrowun
6    6  512     1  Twr > P==>engh BUS BW Twr = 54.41    cacheusd(kbytes)= 28.1      T32s(ms)= 481.7887
mr  nc  nl  numrowun
6    6  512     2  Twr > P==>engh BUS BW Twr = 84.45    cacheusd(kbytes)= 36.1      T32s(ms)= 478.2817
mr  nc  nl  numrowun
6    6  512     4  Twr > P==>engh BUS BW Twr = 117.28    cacheusd(kbytes)= 52.2      T32s(ms)= 476.6129
mr  nc  nl  numrowun
6    6  512     8  Twr > P==>engh BUS BW Twr = 145.40    cacheusd(kbytes)= 84.3      T32s(ms)= 475.9932
mr  nc  nl  numrowun
6    6  512    16  Twr > P==>engh BUS BW Twr = 165.34    cacheusd(kbytes)= 148.4      T32s(ms)= 476.0990
```

```
mr  nc  nl  numrowun
6   6   512   32  Twr > P==>engh BUS BW Twr = 177.52   cacheusd(kbytes)= 276.7     T32s(ms)= 476.9887
mr  nc  nl  numrowun
6   6   512   64  Twr > P==>engh BUS BW Twr = 184.27   cacheusd(kbytes)= 533.3     T32s(ms)= 479.1088
mr  nc  nl  numrowun
6   6   512   128  Twr > P==>engh BUS BW Twr = 187.84   cacheusd(kbytes)= 1046.6     T32s(ms)= 483.5174
mr  nc  nl  numrowun
6   6   512   256  Twr > P==>engh BUS BW Twr = 189.67   cacheusd(kbytes)= 2073.1     T32s(ms)= 483.2032
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun
9   9   32    1  Twr < P ==>btlnck  Twr = 5.4    cacheusd(kbytes)= 40.3     T32s(ms)= 3171.6668
mr  nc  nl  numrowun
9   9   32    2  Twr < P ==>btlnck  Twr = 8.9    cacheusd(kbytes)= 48.3     T32s(ms)= 1906.1309
mr  nc  nl  numrowun
9   9   32    4  Twr < P ==>btlnck  Twr = 13.4    cacheusd(kbytes)= 64.4     T32s(ms)= 1277.8560
mr  nc  nl  numrowun
9   9   32    8  Twr > P==>engh BUS BW Twr = 17.91   cacheusd(kbytes)= 96.5     T32s(ms)= 1137.0935
mr  nc  nl  numrowun
9   9   32    16  Twr > P==>engh BUS BW Twr = 21.50   cacheusd(kbytes)= 160.8     T32s(ms)= 1133.4236
mr  nc  nl  numrowun
9   9   32    32  Twr > P==>engh BUS BW Twr = 23.90   cacheusd(kbytes)= 289.2     T32s(ms)= 1146.3052
mr  nc  nl  numrowun
9   9   32    64  Twr > P==>engh BUS BW Twr = 25.31   cacheusd(kbytes)= 546.2     T32s(ms)= 1179.5732
mr  nc  nl  numrowun
9   9   32    128  Twr > P==>engh BUS BW Twr = 26.08   cacheusd(kbytes)= 1060.2     T32s(ms)= 1249.8616
mr  nc  nl  numrowun
9   9   32    256  Twr > P==>engh BUS BW Twr = 26.48   cacheusd(kbytes)= 2088.2     T32s(ms)= 1244.8586
*********************************************************************************
 mr  nc  nl  numrowun
9   9   64    1  Twr < P ==>btlnck  Twr = 10.7    cacheusd(kbytes)= 40.3     T32s(ms)= 1588.3196
mr  nc  nl  numrowun
9   9   64    2  Twr > P==>engh BUS BW Twr = 17.89   cacheusd(kbytes)= 48.3     T32s(ms)= 1127.0191
mr  nc  nl  numrowun
9   9   64    4  Twr > P==>engh BUS BW Twr = 26.84   cacheusd(kbytes)= 64.4     T32s(ms)= 1106.0744
mr  nc  nl  numrowun
9   9   64    8  Twr > P==>engh BUS BW Twr = 35.81   cacheusd(kbytes)= 96.5     T32s(ms)= 1098.2044
mr  nc  nl  numrowun
9   9   64    16  Twr > P==>engh BUS BW Twr = 43.00   cacheusd(kbytes)= 160.8     T32s(ms)= 1097.6228
mr  nc  nl  numrowun
9   9   64    32  Twr > P==>engh BUS BW Twr = 47.79   cacheusd(kbytes)= 289.2     T32s(ms)= 1104.0388
mr  nc  nl  numrowun
9   9   64    64  Twr > P==>engh BUS BW Twr = 50.62   cacheusd(kbytes)= 546.2     T32s(ms)= 1120.6604
mr  nc  nl  numrowun
9   9   64    128  Twr > P==>engh BUS BW Twr = 52.16   cacheusd(kbytes)= 1060.2     T32s(ms)= 1155.7984
mr  nc  nl  numrowun
9   9   64    256  Twr > P==>engh BUS BW Twr = 52.96   cacheusd(kbytes)= 2088.2     T32s(ms)= 1153.2938
*********************************************************************************
 mr  nc  nl  numrowun
9   9   128    1  Twr > P==>engh BUS BW Twr = 21.46   cacheusd(kbytes)= 40.3     T32s(ms)= 1114.7964
mr  nc  nl  numrowun
9   9   128    2  Twr > P==>engh BUS BW Twr = 35.76   cacheusd(kbytes)= 48.3     T32s(ms)= 1093.8421
mr  nc  nl  numrowun
9   9   128    4  Twr > P==>engh BUS BW Twr = 53.67   cacheusd(kbytes)= 64.4     T32s(ms)= 1084.0970
mr  nc  nl  numrowun
9   9   128    8  Twr > P==>engh BUS BW Twr = 71.61   cacheusd(kbytes)= 96.5     T32s(ms)= 1080.0628
mr  nc  nl  numrowun
9   9   128    16  Twr > P==>engh BUS BW Twr = 85.99   cacheusd(kbytes)= 160.8     T32s(ms)= 1079.7224
mr  nc  nl  numrowun
9   9   128    32  Twr > P==>engh BUS BW Twr = 95.58   cacheusd(kbytes)= 289.2     T32s(ms)= 1082.9056
mr  nc  nl  numrowun
9   9   128    64  Twr > P==>engh BUS BW Twr = 101.23   cacheusd(kbytes)= 546.2     T32s(ms)= 1091.2040
mr  nc  nl  numrowun
9   9   128    128  Twr > P==>engh BUS BW Twr = 104.31   cacheusd(kbytes)= 1060.2     T32s(ms)= 1108.7668
mr  nc  nl  numrowun
9   9   128    256  Twr > P==>engh BUS BW Twr = 105.92   cacheusd(kbytes)= 2088.2     T32s(ms)= 1107.5114
*********************************************************************************
```

```
 mr  nc  nl  numrowun
9    9   256     1  Twr > P==>engh BUS BW Twr = 42.84    cacheusd(kbytes)= 40.3      T32s(ms)= 1088.7601
mr  nc  nl  numrowun
9    9   256     2  Twr > P==>engh BUS BW Twr = 71.47    cacheusd(kbytes)= 48.3      T32s(ms)= 1078.1793
mr  nc  nl  numrowun
9    9   256     4  Twr > P==>engh BUS BW Twr = 107.27    cacheusd(kbytes)= 64.4      T32s(ms)= 1073.1083
mr  nc  nl  numrowun
9    9   256     8  Twr > P==>engh BUS BW Twr = 143.16    cacheusd(kbytes)= 96.5      T32s(ms)= 1070.9920
mr  nc  nl  numrowun
9    9   256    16  Twr > P==>engh BUS BW Twr = 171.93    cacheusd(kbytes)= 160.8      T32s(ms)= 1070.7722
mr  nc  nl  numrowun
9    9   256    32  Twr > P==>engh BUS BW Twr = 191.13    cacheusd(kbytes)= 289.2      T32s(ms)= 1072.3390
mr  nc  nl  numrowun
9    9   256    64  Twr > P==>engh BUS BW Twr = 202.44    cacheusd(kbytes)= 546.2      T32s(ms)= 1076.4758
mr  nc  nl  numrowun
9    9   256   128  Twr > P==>engh BUS BW Twr = 208.61    cacheusd(kbytes)= 1060.2      T32s(ms)= 1085.2510
mr  nc  nl  numrowun
9    9   256   256  Twr > P==>engh BUS BW Twr = 211.84    cacheusd(kbytes)= 2088.2      T32s(ms)= 1084.6202
*********************************************************************************
 mr  nc  nl  numrowun
9    9   512     1  Twr > P==>engh BUS BW Twr = 85.37    cacheusd(kbytes)= 40.3      T32s(ms)= 1076.0548
mr  nc  nl  numrowun
9    9   512     2  Twr > P==>engh BUS BW Twr = 142.50    cacheusd(kbytes)= 48.3      T32s(ms)= 1070.3580
mr  nc  nl  numrowun
9    9   512     4  Twr > P==>engh BUS BW Twr = 214.27    cacheusd(kbytes)= 64.4      T32s(ms)= 1067.6140
mr  nc  nl  numrowun
9    9   512     8  Twr > P==>engh BUS BW Twr = 286.08    cacheusd(kbytes)= 96.5      T32s(ms)= 1066.4566
mr  nc  nl  numrowun
9    9   512    16  Twr > P==>engh BUS BW Twr = 343.67    cacheusd(kbytes)= 160.8      T32s(ms)= 1066.2971
mr  nc  nl  numrowun
9    9   512    32  Twr > P==>engh BUS BW Twr = 382.15    cacheusd(kbytes)= 289.2      T32s(ms)= 1067.0557
mr  nc  nl  numrowun
9    9   512    64  Twr > P==>engh BUS BW Twr = 404.81    cacheusd(kbytes)= 546.2      T32s(ms)= 1069.1117
mr  nc  nl  numrowun
9    9   512   128  Twr > P==>engh BUS BW Twr = 417.18    cacheusd(kbytes)= 1060.2      T32s(ms)= 1073.4931
mr  nc  nl  numrowun
9    9   512   256  Twr > P==>engh BUS BW Twr = 423.66    cacheusd(kbytes)= 2088.2      T32s(ms)= 1073.1746
*********************************************************************************
 *********************************************************************************
```

P=32

```
mr  nc  nl  numrowun  num_loads
2    2   32     1          8192.0  Twr < P ==>btlnck  Twr = 0.9    cacheusd(kbytes)= 12.0      T64s(ms)= 945.1584
mr  nc  nl  numrowun  num_loads
2    2   32     2          4096.0  Twr < P ==>btlnck  Twr = 1.1    cacheusd(kbytes)= 20.0      T64s(ms)= 787.4624
mr  nc  nl  numrowun  num_loads
2    2   32     4          2048.0  Twr < P ==>btlnck  Twr = 1.2    cacheusd(kbytes)= 36.0      T64s(ms)= 708.4608
mr  nc  nl  numrowun  num_loads
2    2   32     8          1024.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 68.0      T64s(ms)= 668.9600
mr  nc  nl  numrowun  num_loads
2    2   32    16           512.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 132.1      T64s(ms)= 649.2096
mr  nc  nl  numrowun  num_loads
2    2   32    32           256.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 260.1      T64s(ms)= 639.3344
mr  nc  nl  numrowun  num_loads
2    2   32    64           128.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 516.3      T64s(ms)= 634.3968
mr  nc  nl  numrowun  num_loads
2    2   32   128            64.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 1028.5      T64s(ms)= 631.9280
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2    2   64     1          8192.0  Twr < P ==>btlnck  Twr = 1.8    cacheusd(kbytes)= 12.0      T64s(ms)= 473.2992
mr  nc  nl  numrowun  num_loads
2    2   64     2          4096.0  Twr < P ==>btlnck  Twr = 2.1    cacheusd(kbytes)= 20.0      T64s(ms)= 393.9392
mr  nc  nl  numrowun  num_loads
2    2   64     4          2048.0  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 36.0      T64s(ms)= 354.4128
mr  nc  nl  numrowun  num_loads
2    2   64     8          1024.0  Twr < P ==>btlnck  Twr = 2.5    cacheusd(kbytes)= 68.0      T64s(ms)= 334.5728
```

```
mr  nc  nl  numrowun  num_loads
2   2   64   16          512.0  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 132.1     T64s(ms)= 324.6528
mr  nc  nl  numrowun  num_loads
2   2   64   32          256.0  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 260.1     T64s(ms)= 319.6928
mr  nc  nl  numrowun  num_loads
2   2   64   64          128.0  Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 516.3     T64s(ms)= 317.2128
mr  nc  nl  numrowun  num_loads
2   2   64   128          64.0  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 1028.5    T64s(ms)= 315.9728
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   128   1          8192.0  Twr < P ==>btlnck  Twr = 3.6    cacheusd(kbytes)= 12.0      T64s(ms)= 237.3696
mr  nc  nl  numrowun  num_loads
2   2   128   2          4096.0  Twr < P ==>btlnck  Twr = 4.3    cacheusd(kbytes)= 20.0      T64s(ms)= 197.3312
mr  nc  nl  numrowun  num_loads
2   2   128   4          2048.0  Twr < P ==>btlnck  Twr = 4.7    cacheusd(kbytes)= 36.0      T64s(ms)= 177.3120
mr  nc  nl  numrowun  num_loads
2   2   128   8          1024.0  Twr < P ==>btlnck  Twr = 5.0    cacheusd(kbytes)= 68.0      T64s(ms)= 167.3792
mr  nc  nl  numrowun  num_loads
2   2   128   16          512.0  Twr < P ==>btlnck  Twr = 5.2    cacheusd(kbytes)= 132.1     T64s(ms)= 162.3744
mr  nc  nl  numrowun  num_loads
2   2   128   32          256.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 260.1     T64s(ms)= 159.8720
mr  nc  nl  numrowun  num_loads
2   2   128   64          128.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 516.3     T64s(ms)= 158.6208
mr  nc  nl  numrowun  num_loads
2   2   128   128          64.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 1028.5    T64s(ms)= 157.9952
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   256   1          8192.0  Twr < P ==>btlnck  Twr = 7.1    cacheusd(kbytes)= 12.0      T64s(ms)= 119.4048
mr  nc  nl  numrowun  num_loads
2   2   256   2          4096.0  Twr < P ==>btlnck  Twr = 8.5    cacheusd(kbytes)= 20.0      T64s(ms)= 99.0272
mr  nc  nl  numrowun  num_loads
2   2   256   4          2048.0  Twr < P ==>btlnck  Twr = 9.5    cacheusd(kbytes)= 36.0      T64s(ms)= 88.8384
mr  nc  nl  numrowun  num_loads
2   2   256   8          1024.0  Twr < P ==>btlnck  Twr = 10.0    cacheusd(kbytes)= 68.0     T64s(ms)= 83.7440
mr  nc  nl  numrowun  num_loads
2   2   256   16          512.0  Twr < P ==>btlnck  Twr = 10.3    cacheusd(kbytes)= 132.1    T64s(ms)= 81.2352
mr  nc  nl  numrowun  num_loads
2   2   256   32          256.0  Twr < P ==>btlnck  Twr = 10.5    cacheusd(kbytes)= 260.1    T64s(ms)= 79.9616
mr  nc  nl  numrowun  num_loads
2   2   256   64          128.0  Twr < P ==>btlnck  Twr = 10.6    cacheusd(kbytes)= 516.3    T64s(ms)= 79.3248
mr  nc  nl  numrowun  num_loads
2   2   256   128          64.0  Twr < P ==>btlnck  Twr = 10.6    cacheusd(kbytes)= 1028.5    T64s(ms)= 79.0064
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   512   1          8192.0  Twr < P ==>btlnck  Twr = 14.0    cacheusd(kbytes)= 12.0      T64s(ms)= 60.4741
mr  nc  nl  numrowun  num_loads
2   2   512   2          4096.0  Twr < P ==>btlnck  Twr = 16.9    cacheusd(kbytes)= 20.0      T64s(ms)= 49.9830
mr  nc  nl  numrowun  num_loads
2   2   512   4          2048.0  Twr < P ==>btlnck  Twr = 18.9    cacheusd(kbytes)= 36.0      T64s(ms)= 44.8217
mr  nc  nl  numrowun  num_loads
2   2   512   8          1024.0  Twr < P ==>btlnck  Twr = 20.0    cacheusd(kbytes)= 68.0      T64s(ms)= 42.4095
mr  nc  nl  numrowun  num_loads
2   2   512   16          512.0  Twr < P ==>btlnck  Twr = 20.7    cacheusd(kbytes)= 132.1     T64s(ms)= 41.5403
mr  nc  nl  numrowun  num_loads
2   2   512   32          256.0  Twr < P ==>btlnck  Twr = 21.0    cacheusd(kbytes)= 260.1     T64s(ms)= 41.7987
mr  nc  nl  numrowun  num_loads
2   2   512   64          128.0  Twr < P ==>btlnck  Twr = 21.2    cacheusd(kbytes)= 516.3     T64s(ms)= 43.2659
mr  nc  nl  numrowun  num_loads
2   2   512   128          64.0  Twr < P ==>btlnck  Twr = 21.3    cacheusd(kbytes)= 1028.5    T64s(ms)= 43.1034
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   32   1          8192.0  Twr < P ==>btlnck  Twr = 1.5    cacheusd(kbytes)= 16.0      T64s(ms)= 1260.9600
mr  nc  nl  numrowun  num_loads
3   3   32   2          4096.0  Twr < P ==>btlnck  Twr = 2.0    cacheusd(kbytes)= 24.0      T64s(ms)= 945.3632
mr  nc  nl  numrowun  num_loads
3   3   32   4          2048.0  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 40.0      T64s(ms)= 787.5648
```

```
mr  nc  nl  numrowun  num_loads
3   3   32    8        1024.0 Twr < P ==>btlnck  Twr = 2.7   cacheusd(kbytes)= 72.1    T64s(ms)= 708.6656
mr  nc  nl  numrowun  num_loads
3   3   32   16         512.0 Twr < P ==>btlnck  Twr = 2.8   cacheusd(kbytes)= 136.1    T64s(ms)= 669.2160
mr  nc  nl  numrowun  num_loads
3   3   32   32         256.0 Twr < P ==>btlnck  Twr = 2.9   cacheusd(kbytes)= 264.3    T64s(ms)= 649.4912
mr  nc  nl  numrowun  num_loads
3   3   32   64         128.0 Twr < P ==>btlnck  Twr = 3.0   cacheusd(kbytes)= 520.5    T64s(ms)= 639.6288
mr  nc  nl  numrowun  num_loads
3   3   32  128          64.0 Twr < P ==>btlnck  Twr = 3.0   cacheusd(kbytes)= 1033.0    T64s(ms)= 634.6976
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   64    1        8192.0 Twr < P ==>btlnck  Twr = 3.0   cacheusd(kbytes)= 16.0    T64s(ms)= 631.2000
mr  nc  nl  numrowun  num_loads
3   3   64    2        4096.0 Twr < P ==>btlnck  Twr = 4.0   cacheusd(kbytes)= 24.0    T64s(ms)= 472.8896
mr  nc  nl  numrowun  num_loads
3   3   64    4        2048.0 Twr < P ==>btlnck  Twr = 4.8   cacheusd(kbytes)= 40.0    T64s(ms)= 393.8880
mr  nc  nl  numrowun  num_loads
3   3   64    8        1024.0 Twr < P ==>btlnck  Twr = 5.3   cacheusd(kbytes)= 72.1    T64s(ms)= 354.3872
mr  nc  nl  numrowun  num_loads
3   3   64   16         512.0 Twr < P ==>btlnck  Twr = 5.6   cacheusd(kbytes)= 136.1    T64s(ms)= 334.6368
mr  nc  nl  numrowun  num_loads
3   3   64   32         256.0 Twr < P ==>btlnck  Twr = 5.8   cacheusd(kbytes)= 264.3    T64s(ms)= 324.7616
mr  nc  nl  numrowun  num_loads
3   3   64   64         128.0 Twr < P ==>btlnck  Twr = 5.9   cacheusd(kbytes)= 520.5    T64s(ms)= 319.8240
mr  nc  nl  numrowun  num_loads
3   3   64  128          64.0 Twr < P ==>btlnck  Twr = 5.9   cacheusd(kbytes)= 1033.0    T64s(ms)= 317.3552
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  128    1        8192.0 Twr < P ==>btlnck  Twr = 6.0   cacheusd(kbytes)= 16.0    T64s(ms)= 316.0128
mr  nc  nl  numrowun  num_loads
3   3  128    2        4096.0 Twr < P ==>btlnck  Twr = 8.0   cacheusd(kbytes)= 24.0    T64s(ms)= 236.6528
mr  nc  nl  numrowun  num_loads
3   3  128    4        2048.0 Twr < P ==>btlnck  Twr = 9.6   cacheusd(kbytes)= 40.0    T64s(ms)= 197.1264
mr  nc  nl  numrowun  num_loads
3   3  128    8        1024.0 Twr < P ==>btlnck  Twr = 10.7   cacheusd(kbytes)= 72.1    T64s(ms)= 177.2864
mr  nc  nl  numrowun  num_loads
3   3  128   16         512.0 Twr < P ==>btlnck  Twr = 11.3   cacheusd(kbytes)= 136.1    T64s(ms)= 167.3664
mr  nc  nl  numrowun  num_loads
3   3  128   32         256.0 Twr < P ==>btlnck  Twr = 11.6   cacheusd(kbytes)= 264.3    T64s(ms)= 162.4064
mr  nc  nl  numrowun  num_loads
3   3  128   64         128.0 Twr < P ==>btlnck  Twr = 11.8   cacheusd(kbytes)= 520.5    T64s(ms)= 159.9264
mr  nc  nl  numrowun  num_loads
3   3  128  128          64.0 Twr < P ==>btlnck  Twr = 11.9   cacheusd(kbytes)= 1033.0    T64s(ms)= 158.6864
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  256    1        8192.0 Twr < P ==>btlnck  Twr = 12.0   cacheusd(kbytes)= 16.0    T64s(ms)= 158.8855
mr  nc  nl  numrowun  num_loads
3   3  256    2        4096.0 Twr < P ==>btlnck  Twr = 15.9   cacheusd(kbytes)= 24.0    T64s(ms)= 119.0105
mr  nc  nl  numrowun  num_loads
3   3  256    4        2048.0 Twr < P ==>btlnck  Twr = 19.2   cacheusd(kbytes)= 40.0    T64s(ms)= 99.3180
mr  nc  nl  numrowun  num_loads
3   3  256    8        1024.0 Twr < P ==>btlnck  Twr = 21.3   cacheusd(kbytes)= 72.1    T64s(ms)= 90.0386
mr  nc  nl  numrowun  num_loads
3   3  256   16         512.0 Twr < P ==>btlnck  Twr = 22.6   cacheusd(kbytes)= 136.1    T64s(ms)= 86.3406
mr  nc  nl  numrowun  num_loads
3   3  256   32         256.0 Twr < P ==>btlnck  Twr = 23.2   cacheusd(kbytes)= 264.3    T64s(ms)= 86.4518
mr  nc  nl  numrowun  num_loads
3   3  256   64         128.0 Twr < P ==>btlnck  Twr = 23.6   cacheusd(kbytes)= 520.5    T64s(ms)= 90.4278
mr  nc  nl  numrowun  num_loads
3   3  256  128          64.0 Twr < P ==>btlnck  Twr = 23.8   cacheusd(kbytes)= 1033.0    T64s(ms)= 89.8043
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  512    1        8192.0 Twr < P ==>btlnck  Twr = 23.7   cacheusd(kbytes)= 16.0    T64s(ms)= 80.3911
mr  nc  nl  numrowun  num_loads
3   3  512    2        4096.0 Twr < P ==>btlnck  Twr = 31.7   cacheusd(kbytes)= 24.0    T64s(ms)= 61.6163
mr  nc  nl  numrowun  num_loads
```

```
3   3   512   4          2048.0  Twr > P==>engh BUS BW Twr = 38.18   cacheusd(kbytes)= 40.0     T64s(ms)= 61.3602
mr  nc  nl  numrowun  num_loads
3   3   512   8          1024.0  Twr > P==>engh BUS BW Twr = 42.54   cacheusd(kbytes)= 72.1     T64s(ms)= 61.7482
mr  nc  nl  numrowun  num_loads
3   3   512   16          512.0  Twr > P==>engh BUS BW Twr = 45.07   cacheusd(kbytes)= 136.1    T64s(ms)= 62.8241
mr  nc  nl  numrowun  num_loads
3   3   512   32          256.0  Twr > P==>engh BUS BW Twr = 46.47   cacheusd(kbytes)= 264.3    T64s(ms)= 65.1193
mr  nc  nl  numrowun  num_loads
3   3   512   64          128.0  Twr > P==>engh BUS BW Twr = 47.21   cacheusd(kbytes)= 520.5    T64s(ms)= 69.7839
mr  nc  nl  numrowun  num_loads
3   3   512   128          64.0  Twr > P==>engh BUS BW Twr = 47.59   cacheusd(kbytes)= 1033.0   T64s(ms)= 69.6270
*********************************************************************************
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   32    1         8192.0  Twr < P ==>btlnck  Twr = 3.4    cacheusd(kbytes)= 28.1     T64s(ms)= 2212.0512
mr  nc  nl  numrowun  num_loads
6   6   32    2         4096.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 36.1     T64s(ms)= 1421.5232
mr  nc  nl  numrowun  num_loads
6   6   32    4         2048.0  Twr < P ==>btlnck  Twr = 7.4    cacheusd(kbytes)= 52.2     T64s(ms)= 1026.1056
mr  nc  nl  numrowun  num_loads
6   6   32    8         1024.0  Twr < P ==>btlnck  Twr = 9.1    cacheusd(kbytes)= 84.3     T64s(ms)= 828.3968
mr  nc  nl  numrowun  num_loads
6   6   32    16         512.0  Twr < P ==>btlnck  Twr = 10.4   cacheusd(kbytes)= 148.4    T64s(ms)= 729.5424
mr  nc  nl  numrowun  num_loads
6   6   32    32         256.0  Twr < P ==>btlnck  Twr = 11.1   cacheusd(kbytes)= 276.7    T64s(ms)= 680.1152
mr  nc  nl  numrowun  num_loads
6   6   32    64         128.0  Twr < P ==>btlnck  Twr = 11.5   cacheusd(kbytes)= 533.3    T64s(ms)= 655.4016
mr  nc  nl  numrowun  num_loads
6   6   32    128         64.0  Twr < P ==>btlnck  Twr = 11.7   cacheusd(kbytes)= 1046.6   T64s(ms)= 643.0448
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   64    1         8192.0  Twr < P ==>btlnck  Twr = 6.8    cacheusd(kbytes)= 28.1     T64s(ms)= 1107.3957
mr  nc  nl  numrowun  num_loads
6   6   64    2         4096.0  Twr < P ==>btlnck  Twr = 10.6   cacheusd(kbytes)= 36.1     T64s(ms)= 712.2726
mr  nc  nl  numrowun  num_loads
6   6   64    4         2048.0  Twr < P ==>btlnck  Twr = 14.7   cacheusd(kbytes)= 52.2     T64s(ms)= 515.8441
mr  nc  nl  numrowun  num_loads
6   6   64    8         1024.0  Twr < P ==>btlnck  Twr = 18.2   cacheusd(kbytes)= 84.3     T64s(ms)= 419.5119
mr  nc  nl  numrowun  num_loads
6   6   64    16         512.0  Twr < P ==>btlnck  Twr = 20.7   cacheusd(kbytes)= 148.4    T64s(ms)= 375.2635
mr  nc  nl  numrowun  num_loads
6   6   64    32         256.0  Twr < P ==>btlnck  Twr = 22.2   cacheusd(kbytes)= 276.7    T64s(ms)= 360.9747
mr  nc  nl  numrowun  num_loads
6   6   64    64         128.0  Twr < P ==>btlnck  Twr = 23.0   cacheusd(kbytes)= 533.3    T64s(ms)= 369.5011
mr  nc  nl  numrowun  num_loads
6   6   64    128         64.0  Twr < P ==>btlnck  Twr = 23.5   cacheusd(kbytes)= 1046.6   T64s(ms)= 363.3186
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   128   1         8192.0  Twr < P ==>btlnck  Twr = 13.7   cacheusd(kbytes)= 28.1     T64s(ms)= 555.0309
mr  nc  nl  numrowun  num_loads
6   6   128   2         4096.0  Twr < P ==>btlnck  Twr = 21.2   cacheusd(kbytes)= 36.1     T64s(ms)= 358.3398
mr  nc  nl  numrowun  num_loads
6   6   128   4         2048.0  Twr < P ==>btlnck  Twr = 29.4   cacheusd(kbytes)= 52.2     T64s(ms)= 261.7129
mr  nc  nl  numrowun  num_loads
6   6   128   8         1024.0  Twr > P==>engh BUS BW Twr = 36.44   cacheusd(kbytes)= 84.3     T64s(ms)= 249.1686
mr  nc  nl  numrowun  num_loads
6   6   128   16         512.0  Twr > P==>engh BUS BW Twr = 41.39   cacheusd(kbytes)= 148.4    T64s(ms)= 252.5243
mr  nc  nl  numrowun  num_loads
6   6   128   32         256.0  Twr > P==>engh BUS BW Twr = 44.40   cacheusd(kbytes)= 276.7    T64s(ms)= 261.5142
mr  nc  nl  numrowun  num_loads
6   6   128   64         128.0  Twr > P==>engh BUS BW Twr = 46.08   cacheusd(kbytes)= 533.3    T64s(ms)= 280.0973
mr  nc  nl  numrowun  num_loads
6   6   128   128         64.0  Twr > P==>engh BUS BW Twr = 46.97   cacheusd(kbytes)= 1046.6   T64s(ms)= 279.4724
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   256   1         8192.0  Twr < P ==>btlnck  Twr = 27.3   cacheusd(kbytes)= 28.1     T64s(ms)= 278.8485
mr  nc  nl  numrowun  num_loads
```

```
6   6  256   2        4096.0 Twr > P==>engh BUS BW Twr = 42.39   cacheusd(kbytes)= 36.1     T64s(ms)= 243.3672
mr  nc  nl  numrowun  num_loads
6   6  256   4        2048.0 Twr > P==>engh BUS BW Twr = 58.81   cacheusd(kbytes)= 52.2     T64s(ms)= 241.9839
mr  nc  nl  numrowun  num_loads
6   6  256   8        1024.0 Twr > P==>engh BUS BW Twr = 72.82   cacheusd(kbytes)= 84.3     T64s(ms)= 242.3260
mr  nc  nl  numrowun  num_loads
6   6  256   16        512.0 Twr > P==>engh BUS BW Twr = 82.75   cacheusd(kbytes)= 148.4    T64s(ms)= 244.2537
mr  nc  nl  numrowun  num_loads
6   6  256   32        256.0 Twr > P==>engh BUS BW Twr = 88.79   cacheusd(kbytes)= 276.7    T64s(ms)= 248.7408
mr  nc  nl  numrowun  num_loads
6   6  256   64        128.0 Twr > P==>engh BUS BW Twr = 92.15   cacheusd(kbytes)= 533.3    T64s(ms)= 258.0284
mr  nc  nl  numrowun  num_loads
6   6  256   128        64.0 Twr > P==>engh BUS BW Twr = 93.93   cacheusd(kbytes)= 1046.6    T64s(ms)= 257.7140
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6  512   1        8192.0 Twr > P==>engh BUS BW Twr = 54.41   cacheusd(kbytes)= 28.1     T64s(ms)= 241.2858
mr  nc  nl  numrowun  num_loads
6   6  512   2        4096.0 Twr > P==>engh BUS BW Twr = 84.45   cacheusd(kbytes)= 36.1     T64s(ms)= 239.6457
mr  nc  nl  numrowun  num_loads
6   6  512   4        2048.0 Twr > P==>engh BUS BW Twr = 117.28   cacheusd(kbytes)= 52.2     T64s(ms)= 239.0345
mr  nc  nl  numrowun  num_loads
6   6  512   8        1024.0 Twr > P==>engh BUS BW Twr = 145.40   cacheusd(kbytes)= 84.3     T64s(ms)= 239.1727
mr  nc  nl  numrowun  num_loads
6   6  512   16        512.0 Twr > P==>engh BUS BW Twr = 165.34   cacheusd(kbytes)= 148.4    T64s(ms)= 240.1202
mr  nc  nl  numrowun  num_loads
6   6  512   32        256.0 Twr > P==>engh BUS BW Twr = 177.52   cacheusd(kbytes)= 276.7    T64s(ms)= 242.3541
mr  nc  nl  numrowun  num_loads
6   6  512   64        128.0 Twr > P==>engh BUS BW Twr = 184.27   cacheusd(kbytes)= 533.3    T64s(ms)= 246.9939
mr  nc  nl  numrowun  num_loads
6   6  512   128        64.0 Twr > P==>engh BUS BW Twr = 187.84   cacheusd(kbytes)= 1046.6    T64s(ms)= 246.8348
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9  32   1        8192.0 Twr < P ==>btlnck  Twr = 5.4    cacheusd(kbytes)= 40.3     T64s(ms)= 3170.4379
mr  nc  nl  numrowun  num_loads
9   9  32   2        4096.0 Twr < P ==>btlnck  Twr = 8.9    cacheusd(kbytes)= 48.3     T64s(ms)= 1903.6733
mr  nc  nl  numrowun  num_loads
9   9  32   4        2048.0 Twr < P ==>btlnck  Twr = 13.4    cacheusd(kbytes)= 64.4     T64s(ms)= 1272.9408
mr  nc  nl  numrowun  num_loads
9   9  32   8        1024.0 Twr < P ==>btlnck  Twr = 17.9    cacheusd(kbytes)= 96.5     T64s(ms)= 962.8742
mr  nc  nl  numrowun  num_loads
9   9  32   16        512.0 Twr < P ==>btlnck  Twr = 21.5    cacheusd(kbytes)= 160.8     T64s(ms)= 818.4402
mr  nc  nl  numrowun  num_loads
9   9  32   32        256.0 Twr < P ==>btlnck  Twr = 23.9    cacheusd(kbytes)= 289.2     T64s(ms)= 767.4218
mr  nc  nl  numrowun  num_loads
9   9  32   64        128.0 Twr < P ==>btlnck  Twr = 25.3    cacheusd(kbytes)= 546.2     T64s(ms)= 784.3098
mr  nc  nl  numrowun  num_loads
9   9  32   128        64.0 Twr < P ==>btlnck  Twr = 26.1    cacheusd(kbytes)= 1060.2     T64s(ms)= 764.4893
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9  64   1        8192.0 Twr < P ==>btlnck  Twr = 10.7    cacheusd(kbytes)= 40.3     T64s(ms)= 1587.7051
mr  nc  nl  numrowun  num_loads
9   9  64   2        4096.0 Twr < P ==>btlnck  Twr = 17.9    cacheusd(kbytes)= 48.3     T64s(ms)= 956.1917
mr  nc  nl  numrowun  num_loads
9   9  64   4        2048.0 Twr < P ==>btlnck  Twr = 26.8    cacheusd(kbytes)= 64.4     T64s(ms)= 644.8704
mr  nc  nl  numrowun  num_loads
9   9  64   8        1024.0 Twr > P==>engh BUS BW Twr = 35.81   cacheusd(kbytes)= 96.5     T64s(ms)= 562.7535
mr  nc  nl  numrowun  num_loads
9   9  64   16        512.0 Twr > P==>engh BUS BW Twr = 43.00   cacheusd(kbytes)= 160.8     T64s(ms)= 566.7140
mr  nc  nl  numrowun  num_loads
9   9  64   32        256.0 Twr > P==>engh BUS BW Twr = 47.79   cacheusd(kbytes)= 289.2     T64s(ms)= 584.2620
mr  nc  nl  numrowun  num_loads
9   9  64   64        128.0 Twr > P==>engh BUS BW Twr = 50.62   cacheusd(kbytes)= 546.2     T64s(ms)= 621.2528
mr  nc  nl  numrowun  num_loads
9   9  64   128        64.0 Twr > P==>engh BUS BW Twr = 52.16   cacheusd(kbytes)= 1060.2     T64s(ms)= 619.9962
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
```

```
9   9   128   1          8192.0  Twr < P ==>btlnck  Twr = 21.5    cacheusd(kbytes)= 40.3     T64s(ms)= 796.3387
mr  nc  nl  numrowun  num_loads
9   9   128   2          4096.0  Twr > P==>engh BUS BW Twr = 35.76   cacheusd(kbytes)= 48.3     T64s(ms)= 551.5509
mr  nc  nl  numrowun  num_loads
9   9   128   4          2048.0  Twr > P==>engh BUS BW Twr = 53.67   cacheusd(kbytes)= 64.4     T64s(ms)= 545.6234
mr  nc  nl  numrowun  num_loads
9   9   128   8          1024.0  Twr > P==>engh BUS BW Twr = 71.61   cacheusd(kbytes)= 96.5     T64s(ms)= 545.3988
mr  nc  nl  numrowun  num_loads
9   9   128   16          512.0  Twr > P==>engh BUS BW Twr = 85.99   cacheusd(kbytes)= 160.8    T64s(ms)= 548.8136
mr  nc  nl  numrowun  num_loads
9   9   128   32          256.0  Twr > P==>engh BUS BW Twr = 95.58   cacheusd(kbytes)= 289.2    T64s(ms)= 557.5752
mr  nc  nl  numrowun  num_loads
9   9   128   64          128.0  Twr > P==>engh BUS BW Twr = 101.23  cacheusd(kbytes)= 546.2    T64s(ms)= 576.0644
mr  nc  nl  numrowun  num_loads
9   9   128   128          64.0  Twr > P==>engh BUS BW Twr = 104.31  cacheusd(kbytes)= 1060.2   T64s(ms)= 575.4330
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256   1          8192.0  Twr > P==>engh BUS BW Twr = 42.84   cacheusd(kbytes)= 40.3     T64s(ms)= 546.1721
mr  nc  nl  numrowun  num_loads
9   9   256   2          4096.0  Twr > P==>engh BUS BW Twr = 71.47   cacheusd(kbytes)= 48.3     T64s(ms)= 540.4300
mr  nc  nl  numrowun  num_loads
9   9   256   4          2048.0  Twr > P==>engh BUS BW Twr = 107.27  cacheusd(kbytes)= 64.4     T64s(ms)= 538.3427
mr  nc  nl  numrowun  num_loads
9   9   256   8          1024.0  Twr > P==>engh BUS BW Twr = 143.16  cacheusd(kbytes)= 96.5     T64s(ms)= 538.1808
mr  nc  nl  numrowun  num_loads
9   9   256   16          512.0  Twr > P==>engh BUS BW Twr = 171.93  cacheusd(kbytes)= 160.8    T64s(ms)= 539.8634
mr  nc  nl  numrowun  num_loads
9   9   256   32          256.0  Twr > P==>engh BUS BW Twr = 191.13  cacheusd(kbytes)= 289.2    T64s(ms)= 544.2318
mr  nc  nl  numrowun  num_loads
9   9   256   64          128.0  Twr > P==>engh BUS BW Twr = 202.44  cacheusd(kbytes)= 546.2    T64s(ms)= 553.4702
mr  nc  nl  numrowun  num_loads
9   9   256   128          64.0  Twr > P==>engh BUS BW Twr = 208.61  cacheusd(kbytes)= 1060.2   T64s(ms)= 553.1514
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   512   1          8192.0  Twr > P==>engh BUS BW Twr = 85.37   cacheusd(kbytes)= 40.3     T64s(ms)= 538.5879
mr  nc  nl  numrowun  num_loads
9   9   512   2          4096.0  Twr > P==>engh BUS BW Twr = 142.50  cacheusd(kbytes)= 48.3     T64s(ms)= 535.8511
mr  nc  nl  numrowun  num_loads
9   9   512   4          2048.0  Twr > P==>engh BUS BW Twr = 214.27  cacheusd(kbytes)= 64.4     T64s(ms)= 534.7023
mr  nc  nl  numrowun  num_loads
9   9   512   8          1024.0  Twr > P==>engh BUS BW Twr = 286.08  cacheusd(kbytes)= 96.5     T64s(ms)= 534.5718
mr  nc  nl  numrowun  num_loads
9   9   512   16          512.0  Twr > P==>engh BUS BW Twr = 343.67  cacheusd(kbytes)= 160.8    T64s(ms)= 535.3883
mr  nc  nl  numrowun  num_loads
9   9   512   32          256.0  Twr > P==>engh BUS BW Twr = 382.15  cacheusd(kbytes)= 289.2    T64s(ms)= 537.5601
mr  nc  nl  numrowun  num_loads
9   9   512   64          128.0  Twr > P==>engh BUS BW Twr = 404.81  cacheusd(kbytes)= 546.2    T64s(ms)= 542.1731
mr  nc  nl  numrowun  num_loads
9   9   512   128          64.0  Twr > P==>engh BUS BW Twr = 417.18  cacheusd(kbytes)= 1060.2   T64s(ms)= 542.0106
*****************************************************************************
 ****************************************************************************
```

P=64

```
mr  nc  nl  numrowun  num_loads
2   2   32    1          8192.0  Twr < P ==>btlnck  Twr = 0.9    cacheusd(kbytes)= 12.0     T128s(ms)= 945.1648
mr  nc  nl  numrowun  num_loads
2   2   32    2          4096.0  Twr < P ==>btlnck  Twr = 1.1    cacheusd(kbytes)= 20.0     T128s(ms)= 787.4688
mr  nc  nl  numrowun  num_loads
2   2   32    4          2048.0  Twr < P ==>btlnck  Twr = 1.2    cacheusd(kbytes)= 36.0     T128s(ms)= 708.4672
mr  nc  nl  numrowun  num_loads
2   2   32    8          1024.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 68.0     T128s(ms)= 668.9664
mr  nc  nl  numrowun  num_loads
2   2   32    16          512.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 132.1    T128s(ms)= 649.2160
mr  nc  nl  numrowun  num_loads
2   2   32    32          256.0  Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 260.1    T128s(ms)= 639.3408
mr  nc  nl  numrowun  num_loads
```

```
2   2   32   64          128.0 Twr < P ==>btlnck  Twr = 1.3    cacheusd(kbytes)= 516.3     T128s(ms)= 634.4032
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   64    1          8192.0 Twr < P ==>btlnck  Twr = 1.8    cacheusd(kbytes)= 12.0      T128s(ms)= 473.3056
mr  nc  nl  numrowun  num_loads
2   2   64    2          4096.0 Twr < P ==>btlnck  Twr = 2.1    cacheusd(kbytes)= 20.0      T128s(ms)= 393.9456
mr  nc  nl  numrowun  num_loads
2   2   64    4          2048.0 Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 36.0      T128s(ms)= 354.4192
mr  nc  nl  numrowun  num_loads
2   2   64    8          1024.0 Twr < P ==>btlnck  Twr = 2.5    cacheusd(kbytes)= 68.0      T128s(ms)= 334.5792
mr  nc  nl  numrowun  num_loads
2   2   64    16          512.0 Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 132.1     T128s(ms)= 324.6592
mr  nc  nl  numrowun  num_loads
2   2   64    32          256.0 Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 260.1     T128s(ms)= 319.6992
mr  nc  nl  numrowun  num_loads
2   2   64    64          128.0 Twr < P ==>btlnck  Twr = 2.6    cacheusd(kbytes)= 516.3     T128s(ms)= 317.2192
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   128   1          8192.0 Twr < P ==>btlnck  Twr = 3.6    cacheusd(kbytes)= 12.0      T128s(ms)= 237.3760
mr  nc  nl  numrowun  num_loads
2   2   128   2          4096.0 Twr < P ==>btlnck  Twr = 4.3    cacheusd(kbytes)= 20.0      T128s(ms)= 197.3376
mr  nc  nl  numrowun  num_loads
2   2   128   4          2048.0 Twr < P ==>btlnck  Twr = 4.7    cacheusd(kbytes)= 36.0      T128s(ms)= 177.3184
mr  nc  nl  numrowun  num_loads
2   2   128   8          1024.0 Twr < P ==>btlnck  Twr = 5.0    cacheusd(kbytes)= 68.0      T128s(ms)= 167.3856
mr  nc  nl  numrowun  num_loads
2   2   128   16          512.0 Twr < P ==>btlnck  Twr = 5.2    cacheusd(kbytes)= 132.1     T128s(ms)= 162.3808
mr  nc  nl  numrowun  num_loads
2   2   128   32          256.0 Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 260.1     T128s(ms)= 159.8784
mr  nc  nl  numrowun  num_loads
2   2   128   64          128.0 Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 516.3     T128s(ms)= 158.6272
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   256   1          8192.0 Twr < P ==>btlnck  Twr = 7.1    cacheusd(kbytes)= 12.0      T128s(ms)= 119.4112
mr  nc  nl  numrowun  num_loads
2   2   256   2          4096.0 Twr < P ==>btlnck  Twr = 8.5    cacheusd(kbytes)= 20.0      T128s(ms)= 99.0336
mr  nc  nl  numrowun  num_loads
2   2   256   4          2048.0 Twr < P ==>btlnck  Twr = 9.5    cacheusd(kbytes)= 36.0      T128s(ms)= 88.8448
mr  nc  nl  numrowun  num_loads
2   2   256   8          1024.0 Twr < P ==>btlnck  Twr = 10.0   cacheusd(kbytes)= 68.0      T128s(ms)= 83.7504
mr  nc  nl  numrowun  num_loads
2   2   256   16          512.0 Twr < P ==>btlnck  Twr = 10.3   cacheusd(kbytes)= 132.1     T128s(ms)= 81.2416
mr  nc  nl  numrowun  num_loads
2   2   256   32          256.0 Twr < P ==>btlnck  Twr = 10.5   cacheusd(kbytes)= 260.1     T128s(ms)= 79.9680
mr  nc  nl  numrowun  num_loads
2   2   256   64          128.0 Twr < P ==>btlnck  Twr = 10.6   cacheusd(kbytes)= 516.3     T128s(ms)= 79.3312
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
2   2   512   1          8192.0 Twr < P ==>btlnck  Twr = 14.0   cacheusd(kbytes)= 12.0      T128s(ms)= 60.4288
mr  nc  nl  numrowun  num_loads
2   2   512   2          4096.0 Twr < P ==>btlnck  Twr = 16.9   cacheusd(kbytes)= 20.0      T128s(ms)= 49.8816
mr  nc  nl  numrowun  num_loads
2   2   512   4          2048.0 Twr < P ==>btlnck  Twr = 18.9   cacheusd(kbytes)= 36.0      T128s(ms)= 44.6080
mr  nc  nl  numrowun  num_loads
2   2   512   8          1024.0 Twr < P ==>btlnck  Twr = 20.0   cacheusd(kbytes)= 68.0      T128s(ms)= 41.9712
mr  nc  nl  numrowun  num_loads
2   2   512   16          512.0 Twr < P ==>btlnck  Twr = 20.7   cacheusd(kbytes)= 132.1     T128s(ms)= 40.6528
mr  nc  nl  numrowun  num_loads
2   2   512   32          256.0 Twr < P ==>btlnck  Twr = 21.0   cacheusd(kbytes)= 260.1     T128s(ms)= 40.0128
mr  nc  nl  numrowun  num_loads
2   2   512   64          128.0 Twr < P ==>btlnck  Twr = 21.2   cacheusd(kbytes)= 516.3     T128s(ms)= 39.6832
*********************************************************************************
 *********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   32   1          8192.0 Twr < P ==>btlnck  Twr = 1.5    cacheusd(kbytes)= 16.0      T128s(ms)= 1260.9664
mr  nc  nl  numrowun  num_loads
3   3   32   2          4096.0 Twr < P ==>btlnck  Twr = 2.0    cacheusd(kbytes)= 24.0      T128s(ms)= 945.3696
```

```
mr  nc  nl  numrowun  num_loads
3   3   32    4        2048.0  Twr < P ==>btlnck  Twr = 2.4    cacheusd(kbytes)= 40.0     T128s(ms)= 787.5712
mr  nc  nl  numrowun  num_loads
3   3   32    8        1024.0  Twr < P ==>btlnck  Twr = 2.7    cacheusd(kbytes)= 72.1     T128s(ms)= 708.6720
mr  nc  nl  numrowun  num_loads
3   3   32   16         512.0  Twr < P ==>btlnck  Twr = 2.8    cacheusd(kbytes)= 136.1    T128s(ms)= 669.2224
mr  nc  nl  numrowun  num_loads
3   3   32   32         256.0  Twr < P ==>btlnck  Twr = 2.9    cacheusd(kbytes)= 264.3    T128s(ms)= 649.4976
mr  nc  nl  numrowun  num_loads
3   3   32   64         128.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 520.5    T128s(ms)= 639.6352
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3   64    1        8192.0  Twr < P ==>btlnck  Twr = 3.0    cacheusd(kbytes)= 16.0     T128s(ms)= 631.2064
mr  nc  nl  numrowun  num_loads
3   3   64    2        4096.0  Twr < P ==>btlnck  Twr = 4.0    cacheusd(kbytes)= 24.0     T128s(ms)= 472.8960
mr  nc  nl  numrowun  num_loads
3   3   64    4        2048.0  Twr < P ==>btlnck  Twr = 4.8    cacheusd(kbytes)= 40.0     T128s(ms)= 393.8944
mr  nc  nl  numrowun  num_loads
3   3   64    8        1024.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 72.1     T128s(ms)= 354.3936
mr  nc  nl  numrowun  num_loads
3   3   64   16         512.0  Twr < P ==>btlnck  Twr = 5.6    cacheusd(kbytes)= 136.1    T128s(ms)= 334.6432
mr  nc  nl  numrowun  num_loads
3   3   64   32         256.0  Twr < P ==>btlnck  Twr = 5.8    cacheusd(kbytes)= 264.3    T128s(ms)= 324.7680
mr  nc  nl  numrowun  num_loads
3   3   64   64         128.0  Twr < P ==>btlnck  Twr = 5.9    cacheusd(kbytes)= 520.5    T128s(ms)= 319.8304
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  128    1        8192.0  Twr < P ==>btlnck  Twr = 6.0    cacheusd(kbytes)= 16.0     T128s(ms)= 316.0192
mr  nc  nl  numrowun  num_loads
3   3  128    2        4096.0  Twr < P ==>btlnck  Twr = 8.0    cacheusd(kbytes)= 24.0     T128s(ms)= 236.6592
mr  nc  nl  numrowun  num_loads
3   3  128    4        2048.0  Twr < P ==>btlnck  Twr = 9.6    cacheusd(kbytes)= 40.0     T128s(ms)= 197.1328
mr  nc  nl  numrowun  num_loads
3   3  128    8        1024.0  Twr < P ==>btlnck  Twr = 10.7   cacheusd(kbytes)= 72.1     T128s(ms)= 177.2928
mr  nc  nl  numrowun  num_loads
3   3  128   16         512.0  Twr < P ==>btlnck  Twr = 11.3   cacheusd(kbytes)= 136.1    T128s(ms)= 167.3728
mr  nc  nl  numrowun  num_loads
3   3  128   32         256.0  Twr < P ==>btlnck  Twr = 11.6   cacheusd(kbytes)= 264.3    T128s(ms)= 162.4128
mr  nc  nl  numrowun  num_loads
3   3  128   64         128.0  Twr < P ==>btlnck  Twr = 11.8   cacheusd(kbytes)= 520.5    T128s(ms)= 159.9328
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  256    1        8192.0  Twr < P ==>btlnck  Twr = 12.0   cacheusd(kbytes)= 16.0     T128s(ms)= 158.7328
mr  nc  nl  numrowun  num_loads
3   3  256    2        4096.0  Twr < P ==>btlnck  Twr = 15.9   cacheusd(kbytes)= 24.0     T128s(ms)= 118.6944
mr  nc  nl  numrowun  num_loads
3   3  256    4        2048.0  Twr < P ==>btlnck  Twr = 19.2   cacheusd(kbytes)= 40.0     T128s(ms)= 98.6752
mr  nc  nl  numrowun  num_loads
3   3  256    8        1024.0  Twr < P ==>btlnck  Twr = 21.3   cacheusd(kbytes)= 72.1     T128s(ms)= 88.7424
mr  nc  nl  numrowun  num_loads
3   3  256   16         512.0  Twr < P ==>btlnck  Twr = 22.6   cacheusd(kbytes)= 136.1    T128s(ms)= 83.7376
mr  nc  nl  numrowun  num_loads
3   3  256   32         256.0  Twr < P ==>btlnck  Twr = 23.2   cacheusd(kbytes)= 264.3    T128s(ms)= 81.2352
mr  nc  nl  numrowun  num_loads
3   3  256   64         128.0  Twr < P ==>btlnck  Twr = 23.6   cacheusd(kbytes)= 520.5    T128s(ms)= 79.9840
*********************************************************************************
 mr  nc  nl  numrowun  num_loads
3   3  512    1        8192.0  Twr < P ==>btlnck  Twr = 23.7   cacheusd(kbytes)= 16.0     T128s(ms)= 80.2375
mr  nc  nl  numrowun  num_loads
3   3  512    2        4096.0  Twr < P ==>btlnck  Twr = 31.7   cacheusd(kbytes)= 24.0     T128s(ms)= 60.0185
mr  nc  nl  numrowun  num_loads
3   3  512    4        2048.0  Twr < P ==>btlnck  Twr = 38.2   cacheusd(kbytes)= 40.0     T128s(ms)= 50.1468
mr  nc  nl  numrowun  num_loads
3   3  512    8        1024.0  Twr < P ==>btlnck  Twr = 42.5   cacheusd(kbytes)= 72.1     T128s(ms)= 45.6866
mr  nc  nl  numrowun  num_loads
3   3  512   16         512.0  Twr < P ==>btlnck  Twr = 45.1   cacheusd(kbytes)= 136.1    T128s(ms)= 44.4462
mr  nc  nl  numrowun  num_loads
```

```
3   3   512   32          256.0  Twr < P ==>btlnck  Twr = 46.5    cacheusd(kbytes)= 264.3     T128s(ms)= 45.7094
mr  nc  nl  numrowun  num_loads
3   3   512   64          128.0  Twr < P ==>btlnck  Twr = 47.2    cacheusd(kbytes)= 520.5     T128s(ms)= 45.0779
*****************************************************************************
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   32    1          8192.0  Twr < P ==>btlnck  Twr = 3.4    cacheusd(kbytes)= 28.1     T128s(ms)= 2212.0576
mr  nc  nl  numrowun  num_loads
6   6   32    2          4096.0  Twr < P ==>btlnck  Twr = 5.3    cacheusd(kbytes)= 36.1     T128s(ms)= 1421.5296
mr  nc  nl  numrowun  num_loads
6   6   32    4          2048.0  Twr < P ==>btlnck  Twr = 7.4    cacheusd(kbytes)= 52.2     T128s(ms)= 1026.1120
mr  nc  nl  numrowun  num_loads
6   6   32    8          1024.0  Twr < P ==>btlnck  Twr = 9.1    cacheusd(kbytes)= 84.3     T128s(ms)= 828.4032
mr  nc  nl  numrowun  num_loads
6   6   32    16          512.0  Twr < P ==>btlnck  Twr = 10.4   cacheusd(kbytes)= 148.4    T128s(ms)= 729.5488
mr  nc  nl  numrowun  num_loads
6   6   32    32          256.0  Twr < P ==>btlnck  Twr = 11.1   cacheusd(kbytes)= 276.7    T128s(ms)= 680.1216
mr  nc  nl  numrowun  num_loads
6   6   32    64          128.0  Twr < P ==>btlnck  Twr = 11.5   cacheusd(kbytes)= 533.3    T128s(ms)= 655.4080
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   64    1          8192.0  Twr < P ==>btlnck  Twr = 6.8    cacheusd(kbytes)= 28.1     T128s(ms)= 1106.7520
mr  nc  nl  numrowun  num_loads
6   6   64    2          4096.0  Twr < P ==>btlnck  Twr = 10.6   cacheusd(kbytes)= 36.1     T128s(ms)= 710.9760
mr  nc  nl  numrowun  num_loads
6   6   64    4          2048.0  Twr < P ==>btlnck  Twr = 14.7   cacheusd(kbytes)= 52.2     T128s(ms)= 513.2416
mr  nc  nl  numrowun  num_loads
6   6   64    8          1024.0  Twr < P ==>btlnck  Twr = 18.2   cacheusd(kbytes)= 84.3     T128s(ms)= 414.2976
mr  nc  nl  numrowun  num_loads
6   6   64    16          512.0  Twr < P ==>btlnck  Twr = 20.7   cacheusd(kbytes)= 148.4    T128s(ms)= 364.8256
mr  nc  nl  numrowun  num_loads
6   6   64    32          256.0  Twr < P ==>btlnck  Twr = 22.2   cacheusd(kbytes)= 276.7    T128s(ms)= 340.0896
mr  nc  nl  numrowun  num_loads
6   6   64    64          128.0  Twr < P ==>btlnck  Twr = 23.0   cacheusd(kbytes)= 533.3    T128s(ms)= 327.7216
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   128   1          8192.0  Twr < P ==>btlnck  Twr = 13.7   cacheusd(kbytes)= 28.1     T128s(ms)= 554.4165
mr  nc  nl  numrowun  num_loads
6   6   128   2          4096.0  Twr < P ==>btlnck  Twr = 21.2   cacheusd(kbytes)= 36.1     T128s(ms)= 357.1110
mr  nc  nl  numrowun  num_loads
6   6   128   4          2048.0  Twr < P ==>btlnck  Twr = 29.4   cacheusd(kbytes)= 52.2     T128s(ms)= 259.2553
mr  nc  nl  numrowun  num_loads
6   6   128   8          1024.0  Twr < P ==>btlnck  Twr = 36.4   cacheusd(kbytes)= 84.3     T128s(ms)= 212.3055
mr  nc  nl  numrowun  num_loads
6   6   128   16          512.0  Twr < P ==>btlnck  Twr = 41.4   cacheusd(kbytes)= 148.4    T128s(ms)= 192.5947
mr  nc  nl  numrowun  num_loads
6   6   128   32          256.0  Twr < P ==>btlnck  Twr = 44.4   cacheusd(kbytes)= 276.7    T128s(ms)= 190.3443
mr  nc  nl  numrowun  num_loads
6   6   128   64          128.0  Twr < P ==>btlnck  Twr = 46.1   cacheusd(kbytes)= 533.3    T128s(ms)= 184.1538
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   256   1          8192.0  Twr < P ==>btlnck  Twr = 27.3   cacheusd(kbytes)= 28.1     T128s(ms)= 278.5413
mr  nc  nl  numrowun  num_loads
6   6   256   2          4096.0  Twr < P ==>btlnck  Twr = 42.4   cacheusd(kbytes)= 36.1     T128s(ms)= 180.7590
mr  nc  nl  numrowun  num_loads
6   6   256   4          2048.0  Twr < P ==>btlnck  Twr = 58.8   cacheusd(kbytes)= 52.2     T128s(ms)= 133.4185
mr  nc  nl  numrowun  num_loads
6   6   256   8          1024.0  Twr > P==>engh BUS BW Twr = 72.82   cacheusd(kbytes)= 84.3     T128s(ms)= 126.5519
mr  nc  nl  numrowun  num_loads
6   6   256   16          512.0  Twr > P==>engh BUS BW Twr = 82.75   cacheusd(kbytes)= 148.4    T128s(ms)= 130.5417
mr  nc  nl  numrowun  num_loads
6   6   256   32          256.0  Twr > P==>engh BUS BW Twr = 88.79   cacheusd(kbytes)= 276.7    T128s(ms)= 140.0605
mr  nc  nl  numrowun  num_loads
6   6   256   64          128.0  Twr > P==>engh BUS BW Twr = 92.15   cacheusd(kbytes)= 533.3    T128s(ms)= 139.9012
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
6   6   512   1          8192.0  Twr < P ==>btlnck  Twr = 54.4   cacheusd(kbytes)= 28.1     T128s(ms)= 140.6037
```

```
mr  nc  nl  numrowun  num_loads
6   6   512   2        4096.0  Twr > P==>engh BUS BW Twr = 84.45   cacheusd(kbytes)= 36.1    T128s(ms)= 121.1722
mr  nc  nl  numrowun  num_loads
6   6   512   4        2048.0  Twr > P==>engh BUS BW Twr = 117.28  cacheusd(kbytes)= 52.2    T128s(ms)= 121.0001
mr  nc  nl  numrowun  num_loads
6   6   512   8        1024.0  Twr > P==>engh BUS BW Twr = 145.40  cacheusd(kbytes)= 84.3    T128s(ms)= 121.9799
mr  nc  nl  numrowun  num_loads
6   6   512   16        512.0  Twr > P==>engh BUS BW Twr = 165.34  cacheusd(kbytes)= 148.4   T128s(ms)= 124.2716
mr  nc  nl  numrowun  num_loads
6   6   512   32        256.0  Twr > P==>engh BUS BW Twr = 177.52  cacheusd(kbytes)= 276.7   T128s(ms)= 129.0244
mr  nc  nl  numrowun  num_loads
6   6   512   64        128.0  Twr > P==>engh BUS BW Twr = 184.27  cacheusd(kbytes)= 533.3   T128s(ms)= 128.9428
*******************************************************************************
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   32    1        8192.0  Twr < P ==>btlnck  Twr = 5.4    cacheusd(kbytes)= 40.3    T128s(ms)= 3168.6784
mr  nc  nl  numrowun  num_loads
9   9   32    2        4096.0  Twr < P ==>btlnck  Twr = 8.9    cacheusd(kbytes)= 48.3    T128s(ms)= 1900.1472
mr  nc  nl  numrowun  num_loads
9   9   32    4        2048.0  Twr < P ==>btlnck  Twr = 13.4   cacheusd(kbytes)= 64.4    T128s(ms)= 1265.8816
mr  nc  nl  numrowun  num_loads
9   9   32    8        1024.0  Twr < P ==>btlnck  Twr = 17.9   cacheusd(kbytes)= 96.5    T128s(ms)= 948.7488
mr  nc  nl  numrowun  num_loads
9   9   32    16        512.0  Twr < P ==>btlnck  Twr = 21.5   cacheusd(kbytes)= 160.8   T128s(ms)= 790.1824
mr  nc  nl  numrowun  num_loads
9   9   32    32        256.0  Twr < P ==>btlnck  Twr = 23.9   cacheusd(kbytes)= 289.2   T128s(ms)= 710.8992
mr  nc  nl  numrowun  num_loads
9   9   32    64        128.0  Twr < P ==>btlnck  Twr = 25.3   cacheusd(kbytes)= 546.2   T128s(ms)= 671.2576
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   64    1        8192.0  Twr < P ==>btlnck  Twr = 10.7   cacheusd(kbytes)= 40.3    T128s(ms)= 1586.4763
mr  nc  nl  numrowun  num_loads
9   9   64    2        4096.0  Twr < P ==>btlnck  Twr = 17.9   cacheusd(kbytes)= 48.3    T128s(ms)= 953.7341
mr  nc  nl  numrowun  num_loads
9   9   64    4        2048.0  Twr < P ==>btlnck  Twr = 26.8   cacheusd(kbytes)= 64.4    T128s(ms)= 639.9552
mr  nc  nl  numrowun  num_loads
9   9   64    8        1024.0  Twr < P ==>btlnck  Twr = 35.8   cacheusd(kbytes)= 96.5    T128s(ms)= 488.2502
mr  nc  nl  numrowun  num_loads
9   9   64    16        512.0  Twr < P ==>btlnck  Twr = 43.0   cacheusd(kbytes)= 160.8   T128s(ms)= 422.7666
mr  nc  nl  numrowun  num_loads
9   9   64    32        256.0  Twr < P ==>btlnck  Twr = 47.8   cacheusd(kbytes)= 289.2   T128s(ms)= 410.7626
mr  nc  nl  numrowun  num_loads
9   9   64    64        128.0  Twr < P ==>btlnck  Twr = 50.6   cacheusd(kbytes)= 546.2   T128s(ms)= 390.9389
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   128   1        8192.0  Twr < P ==>btlnck  Twr = 21.5   cacheusd(kbytes)= 40.3    T128s(ms)= 795.7243
mr  nc  nl  numrowun  num_loads
9   9   128   2        4096.0  Twr < P ==>btlnck  Twr = 35.8   cacheusd(kbytes)= 48.3    T128s(ms)= 481.2221
mr  nc  nl  numrowun  num_loads
9   9   128   4        2048.0  Twr < P ==>btlnck  Twr = 53.7   cacheusd(kbytes)= 64.4    T128s(ms)= 328.3776
mr  nc  nl  numrowun  num_loads
9   9   128   8        1024.0  Twr > P==>engh BUS BW Twr = 71.61   cacheusd(kbytes)= 96.5    T128s(ms)= 286.7075
mr  nc  nl  numrowun  num_loads
9   9   128   16        512.0  Twr > P==>engh BUS BW Twr = 85.99   cacheusd(kbytes)= 160.8   T128s(ms)= 292.6208
mr  nc  nl  numrowun  num_loads
9   9   128   32        256.0  Twr > P==>engh BUS BW Twr = 95.58   cacheusd(kbytes)= 289.2   T128s(ms)= 311.5732
mr  nc  nl  numrowun  num_loads
9   9   128   64        128.0  Twr > P==>engh BUS BW Twr = 101.23  cacheusd(kbytes)= 546.2   T128s(ms)= 311.2538
*******************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   256   1        8192.0  Twr < P ==>btlnck  Twr = 42.8   cacheusd(kbytes)= 40.3    T128s(ms)= 400.6555
mr  nc  nl  numrowun  num_loads
9   9   256   2        4096.0  Twr > P==>engh BUS BW Twr = 71.47   cacheusd(kbytes)= 48.3    T128s(ms)= 274.9552
mr  nc  nl  numrowun  num_loads
9   9   256   4        2048.0  Twr > P==>engh BUS BW Twr = 107.27  cacheusd(kbytes)= 64.4    T128s(ms)= 272.8163
mr  nc  nl  numrowun  num_loads
9   9   256   8        1024.0  Twr > P==>engh BUS BW Twr = 143.16  cacheusd(kbytes)= 96.5    T128s(ms)= 274.5568
```

```
mr  nc  nl  numrowun  num_loads
9   9   256   16          512.0  Twr > P==>engh BUS BW Twr = 171.93   cacheusd(kbytes)= 160.8    T128s(ms)= 279.0410
mr  nc  nl  numrowun  num_loads
9   9   256   32          256.0  Twr > P==>engh BUS BW Twr = 191.13   cacheusd(kbytes)= 289.2    T128s(ms)= 288.5110
mr  nc  nl  numrowun  num_loads
9   9   256   64          128.0  Twr > P==>engh BUS BW Twr = 202.44   cacheusd(kbytes)= 546.2    T128s(ms)= 288.3482
*****************************************************************************
 mr  nc  nl  numrowun  num_loads
9   9   512   1          8192.0  Twr > P==>engh BUS BW Twr = 85.37   cacheusd(kbytes)= 40.3     T128s(ms)= 271.1461
mr  nc  nl  numrowun  num_loads
9   9   512   2          4096.0  Twr > P==>engh BUS BW Twr = 142.50   cacheusd(kbytes)= 48.3     T128s(ms)= 269.2967
mr  nc  nl  numrowun  num_loads
9   9   512   4          2048.0  Twr > P==>engh BUS BW Twr = 214.27   cacheusd(kbytes)= 64.4     T128s(ms)= 269.1759
mr  nc  nl  numrowun  num_loads
9   9   512   8          1024.0  Twr > P==>engh BUS BW Twr = 286.08   cacheusd(kbytes)= 96.5     T128s(ms)= 270.0214
mr  nc  nl  numrowun  num_loads
9   9   512   16          512.0  Twr > P==>engh BUS BW Twr = 343.67   cacheusd(kbytes)= 160.8    T128s(ms)= 272.2511
mr  nc  nl  numrowun  num_loads
9   9   512   32          256.0  Twr > P==>engh BUS BW Twr = 382.15   cacheusd(kbytes)= 289.2    T128s(ms)= 276.9799
mr  nc  nl  numrowun  num_loads
9   9   512   64          128.0  Twr > P==>engh BUS BW Twr = 404.81   cacheusd(kbytes)= 546.2    T128s(ms)= 276.8954
*****************************************************************************
 ****************************************************************************
```

REFERENCES

1. Tien-Fu Chen and Jean-Loup Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Transactions on Computers*, Vol. 44, No. 5, pp. 609-623, May 1995.

2. David J. Lilja, " The Impact of Parallel Loop Scheduling Strategies on Prefetching in a Shared Memory Multiprocessor," *IEEE Transactions on Parallel and Distributed Processing*, Vol. 5, No. 6, pp. 573-584, June 1994.

3. Todd Mowry and Anoop Gupta, "Tolerating Latency Through Software-Controlled Prefetching in Shared-Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 12, pp. 87-106, 1991.

4. Fredrik Dahlgren, Michel Dubois, and Per Stenstrom, "Sequential Hardware Prefetching in Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Processing*, Vol. 6, No. 7, pp. 733-746, July 1995.

5. Chung-Ho Chen and Arun K. Somani, "Effects of Cache Traffic on Shared Bus Multiprocessor Systems," *International Conference on Parallel Processing*, pp. 285-288, 1992.

6. S. Chandra, J. R. Larus, and A. Rogers, "What is Time Spent in Message-Passing and Shared-Memory Programs?," 6th *International Conference on Architectural Support for Programs Langs. and OS's,* October 1994.

7. Shin-ichiro Mori, Hiheki Saito, Masahiro Goshima, Mamoru Yanagihara, Takashi Tanaka, David Fraser, Kazuki Joe, Hiroyuki Nitta, and Shihji Tomita, "A Distributed Shared Memory Multiprocessor: ASURA - Memory and Cache Architectures," *Proceedings of Supercomputing, Portland- Oregon,* pp. 740-749, November 1993.

8. Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative and Prefetch Buffers," *Computer Architecture News,* pp. 364-373, January 1990.

9. Jean-Loup. Baer and T.-F. Chen, "An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty," *Proceedings of Supercomputing'91*, pp. 176-186, November 1991.

10. Ingrid Y. Bucher and Donald A. Calahan, "Models of Access Delays in Multiprocessor Memories," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 3, No. 3, pp. 270-280, May 1992.

11. John K. Bennett, Sandhya Dwarkadas, Jay Greenwood, and Evan Speight, "Willow: A Scalable Shared Memory Multiprocessors," *Proceedings of Supercomputing '92*, pp. 336-345, 1992.

12. Nayeem Islam and Roy H. Campbell, "Design Considerations for Shared Memory Multiprocessor Message Systems," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 3, No. 6, pp. 702-711, November 1992.

13. Arun Nanda and Lionel M. Ni, "MAD Kernels: An Experimental Testbed to Study Multiprocessor Memory System Behavior," *International Conference on Parallel Processing,* pp. 28-35, 1992.

14. Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam, "The Stanford Dash Multiprocessor," *IEEE Computer*, pp. 63-79, March 1992.

15. Analog Devices, Norwood, MA, *ADSP-2106X SHARC^{TM} User's Manual*, 1997.

16. Analog Devices, Norwood, MA, *Digital Signal Processing in VLSI*, 1990.

17. Andrew S. Tanenbaum, M. Frans Kaashoek, Henri E. Bal, "Parallel Programming Using Shared Objects and Broadcasting," *IEEE Computer,* pp. 10-19, November 1992.

18. K. Rajan, K. S. Sangunni, and J. Ramakrishna, "Dual-DSP System for Signal and Image Processing," *Microprocessors and Microsystems,* Vol. 17, No. 9, pp. 556-560, November 1993.

19. Betty Prince, "Memory in the Fast Lane," *IEEE Specrtrum*, pp. 38-41, February 1994.

20. Oxford Micro Devices Inc., "A236 Parallel Video Digital Signal Processor Chip," *Data Sheet Summary of Oxford Micro Devices Inc.*, June 1997.

21. Dave Bursky, "Parallelism Pushes DSP Throughput," *Electronic Design*, pp. 151-154, March 1994.

22. Dave Bursky, "Parallel-Processing DSP Chip Delivers Top Speed," *Electronic Design,* pp. 43-49, October 1991

23. R. C. Gonzales and P. Wintz, *Digital Image Processing*, Addison-Wesley, 1987.

24. P. Danielson and S. Levialdi, "Computer Architectures for Pictorial Information Systems," *IEEE Computer*, pp. 53-57, November 1981.

25. Yasuyuki Okumura, Kazunari Irie, and Ryozo Kishimoto, "Multiprocessor DSP with Multistage Switching Network for Video Coding," *IEEE Transactions on Communications,* Vol. 39, No. 6, pp. 938-946, June 1991.

26. Srinath V. Ramaswamy and Gerald D. Miller, "Multiprocessor DSP Architectures that Implement the FCT Based JPEG Still Picture Image Compression Algorithm with Arithmetic Coding," *IEEE Transactions on Consumer Electronics,* pp. 1-5, October 1992.

27. Marc Duranton, "Image Processing by Neural Networks," *IEEE Micro*, pp. 12-19, October 1996.

28. Erick Hagersten, Anders Landin, and Seif Haridi, "DDM - A Cache-Only Memory Architecture," *IEEE Computer*, pp. 44-54, September 1992.

29. John Brian, "DSP: The Secret is Out," *PC Computing,* April 1994.

30. Roland L. Lee, Pen-Chung Yew, and Duncan H. Lawrie, "Data Prefetching in Shared Memory Multiprocessors," *Proceedings of the International Conference on Parallel Processing,* pp. 28-31, 1987.

31. Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing*, Benjamin Cummings, Redwood City, California, 1994.

32. Kai Hwang, *Advanced Computer Architecture with Parallel Programming,* McGraw Hill, Englewood Cliffs, NJ, 1993.

33. Ted G. Lewis, "Where is Computing Headed," *IEEE Computer*, Vol. 27, No. 8, pp. 59-63 August 1994.

34. Arvind, "Prospects of Ubiquitous Parallel Computing," *Proceedings of Parallel Processing Symposium, 8th Int'l (ICPP 94)*, IEEE CS Press, Los Alamitos, California, Order No. 5602-02U, pp. 2, April 1994.

35. Ted G. Lewis, "Supercomputers Ain't so Super," *IEEE Computer*, pp. 5-8, November 1994.

36. Per Stenstrom, "A Survey of Cache Coherence Scheme for Multiprocessors," *IEEE Computer*, Vol. 23, No. 6, pp. 12-24, June 1990.

37. Kevin Dowd, *High Performance Computing*, O'Reilly&Associates Inc., California, 1993.

38. Chang J. H., Ibarra O., Pong T.C., and Sohn S., "Convolution on a Pyramid Computer," *Proceedings of the International Conference on Parallel Processing,* pp. 780-782, 1987

39. Ranka S. and Sanhi S., "Convolution on SIMD Mesh Connected Multicomputers," *Proceedings of the International Conference on Parallel Processing,* pp. 212-216, 1987

40. Prasanna Kumar V. and Krishnan V., "Efficient Image Template Matching on SIMD Hypercube Machines," *Proceedings of the International Conference on Parallel Processing,* pp. 765-771, 1987

41. Ranka S. and Sanhi S., "Image Template Matching on MIMD Hypercube Multicomputer," *Proceedings of the International Conference on Parallel Processing,* pp. 92-99, 1988